ほんとうのコンピュータと出逢うために

はじめて読む マシン語

村瀬 康治 著



アスキー出版局

ほんとうのコンピュータと出逢うために

はじめて読む
マシン語

村瀬康治 著

アスキー出版局

はじめて読む
マシン語

村瀬　康治　著

●

アスキー出版局

●本書を読む前に

●本書ではZ-80という8ビットCPUを搭載しているパーソナル・コンピュータを通して，マシン語の学習を行ないます．Z-80（または，これとコンパチブルのμPD780）は多くのパーソナル・コンピュータで採用されており，マシン語をはじめて学ぶのにも最適なCPUといえるでしょう．

マシン語を学ぶ道具としては，「モニタ」と呼ばれるプログラムをBASICで作成し，これを利用します．本文で示したモニタ・プログラムのリストはPC-8801のN-BASICで記述されていますが,PC-8001，PC-8001 MK IIでも変更なしで使用できます．また，なるべく多くのZ-80マシン上で本書が利用できるように，巻末のAPPENDIXでは他の機種についての対策もまとめてありますので，多くの人に本書を利用していただけることと思います．

はじめに

本書のタイトルは「はじめて読むマシン語」ですが，けっしてマシン語だけの本ではありません．マシン語の命令はBASIC言語の命令とは違って，コンピュータの働きの中心であるCPU(Z-80や8080,8085など)を直接コントロールします．よってマシン語を学ぶにはまず，コンピュータの仕組みはどうなっているのか，また，CPUを中心にそれらはどのような働きをするのかを知らなければなりません．この基礎知識があってこそマシン語の理解が可能なのです．

今，世の中には多くのマシン語入門のための良書があります．しかしマシン語そのものを学習する以前の，コンピュータの基礎知識を，わかりやすく解説してあるものは少ないようです．私が本書を書こうと思った理由もここにあります．今まで，マシン語を学ぼうと思って参考書を手にしても，どうも中にはいっていけなかった人にこそ読んでいただきたいと思います．

本書はマシン語の命令のすべてを解説しているわけではなく，またマシン語のプログラミング技術の多くを教えているわけでもありません．本書はまず，マシン語とは何か，マシン語のプログラムとはどういうものか，それはどのように作ってどう実行されるのか，CPUとは何か，コンピュータ全体はどう働くのか，BASIC言語を実行するときとは何が違うのか，などの，マシン語命令を学ぶ以前に必要となるコンピュータの基礎知識をわかりやすく解説します．そのあとで，主要な命令のいくつかを，やさしい実習プログラムを基に解説していきます．

本書で実習解説する命令は，いずれもコンピュータを働かせるための基本的な命令であり，プログラミング上なくてはならない重要な命令です．

本書を読み終える頃には，今までBASICは使えても，ブラックボックスであったコンピュータの中身が見えるようになっていることでしょう．そうなればあなたはコンピュータに対して自立したと言ってもよいのです．あなたは自分の目で良い参考書，適した参考書を捜し出すだけの知識がついているはずです．アセンブラなり各種言語なり，いろいろな分野に挑戦していくだけの基礎もできているでしょう．また今まで使っていたBASIC言語も，新たな観点から見直すこともできるはずです．コンピュータ・アーキテクチャの基礎を学ぶことこそ何よりも必要なことであり，本書の目的もそこにあるのです．

本書を最初から順にしっかりと読んでいってください．しっかり読んでいけば，コンピュータ・アーキテクチャの一つひとつは実に単純であることに気付くはずです．

BASICの殻から一歩を踏み出し，マシン語——つまりはコンピュータの基本を学ぼうとしている賢明な読者に，心から励ましの言葉を送ります．

あなたと〝コンピュータ〟，そのほんとうの出逢いは本書から始まるのかも知れません．

1983年9月

著者　村瀬康治

CONTENTS

序.ほんとうのコンピュータと出逢うために

BASICを離れて――6

マシン語を学ぼう――9

1.マシン語を学ぶ道具 MCモニタ

1.1 MCモニタの作成――13

1.2 MCモニタの使い方――17

Dコマンド(メモリダンプ)――19

Sコマンド(メモリ内容の変更)――22

Gコマンド(プログラムの実行)――24

2.メモリに関する基礎知識

2.1 アドレス,バイト,ビット――29

アドレス(Address)――30

バイト(Byte)――31

ビット(Bit)――31

2.2 メモリに記憶されているデータを見る――33

3. 2進法と16進法

3.1 16進の予備知識――41

3.2 2進,8ビット,16進の関連――43

4.コンピュータに直接命令できるのはマシン語だけ

4.1 BASICはマシン語のプログラム――53

4.2 マシン語でコンピュータを直接働かせてみる――57

5.CPU,メモリ,I/O,バス

5.1 コンピュータのシステム構成――63

5.2 CPUの働き――65

メモリのリード/ライト――65

5.3 メモリの種類と構成――67

ROMとRAM――67

バンク切換え――68

5.4 I/Oの選択メカニズム――70

5.5 バスの概念――72

6.ニーモニックとアセンブラ

6.1 ニーモニックとアセンブリ言語――77

6.2 ハンド・アセンブルとアセンブラ――80

ハンド・アセンブル――80

本格的なアセンブラの例――85

7.コンピュータの中心CPU

7.1　プログラム実行のメカニズム——93
プログラム・カウンタ——96
7.2　レジスタとその働き——98
メイン・レジスタ群——99
フラグ・レジスタ——99
裏レジスタ群——100
専用レジスタ群——100

8.マシン語命令の実習

8.1　マシン語実習の前に——103
8.2　8ビット・ロード命令——106
8.3　8ビット算術演算命令——119
比較(コンペア)命令——134
8.4　各種フラグと条件分岐——136
フラグ——136
条件分岐——138
8.5　ジャンプ命令——141
相対(リラティブ)アドレス・ジャンプ——150
8.6　コール,リターン命令——153
8.7　スタックの概念とスタック・ポインタ——161
8.8　ローテート,シフト命令——168
ローテート命令——170
シフト命令——171
8.9　16ビット・ロード命令——176
8.10　16ビット算術演算命令——184
8.11　入出力命令——191
8.12　その他の命令——195
NOPとHALT命令——195
ブロック転送命令——197

9.やさしいプログラミング例

9.1　ブロック転送プログラム——201
9.2　スクリーンへのキャラクタ表示——203
9.3　ブザーの遊び——208
9.4　プリンタを動かすプログラム——210

終．マシン語から広がる世界

APPENDIX

1　各機種でMCモニタを利用する場合の注意点——223
2　本書で使ったROM内ルーチンの機種別アドレス——228
3　機能別主要命令表——230
4　キャラクタコード表(PC-8001/8801用)——236
索引——237

Illustration　Yohichiro Den

序章

BASICを離れて

豊富な機能を持ちたいへん使いやすいBASICという言語がありながら，なぜ今からマシン語などをやらなければならないのか，その話から始めなくてはなりません．

それは，みなさんが使っているパーソナル・コンピュータは，BASIC専用のBASICマシンではなく，コンピュータであるという事実から出発します．

私が最初に経験したBASICは，プロセッサー・テクノロジー社(米国)のSOL-20というパーソナル・コンピュータに付属していた，スクリーン・エディタ内蔵！(今では当り前ですが)の8K-BASICでした．1977年の秋頃の話です．その後いろいろなBASICを使いましたが主なものを挙げてみると，次のようなものです．

テープベース	8K-BASIC	上記のもの．テープベースながら FILE#, CLOSE #, PRINT#, READ#などのファイル操作が可能であった．
	パロアルト TINY BASIC	1977年8月号から数カ月，ASCII誌に掲載されたものをSOL-20に移植したもの．
	15K-拡張BASIC	プロセッサー・テクノロジー社の8K-BASICの次の製品．強力なマトリックス操作のできる「MAT」ステートメントがあった．
CP/Mディスクベース	MBASIC	マイクロソフト社．NEC PC-8001 N-BASICのオリジナル版．もちろんグラフィック関係のコマンドはない．
	BASCOM (MBASICのコンパイラ版)	マイクロソフト社．最初のコンパイラ形BASIC．
	CBASIC	デジタルリサーチ社．アメリカでビジネス用に広く普及している．
	CB-80 (CBASICのコンパイラ版)	デジタルリサーチ社．行番号は一切不用．ある程度の構造化プログラミングが行なえる．

どれもそれぞれの特徴があり，とてもよくできたBASICだと思っています．CP／M（シーピーエム）ベースのBASICは，SOL-20上でもたまに使用しますが，現在は主にPC-8001や8801のCP／M上で走らせています．

これらのBASICの中で，CBASICのコンパイラ版であるCB-80は，マイクロソフト系BASICとはちょっと違ったBASICですので，参考までにその簡単なソース・プログラムを示しておきましょう．

```
        ¥  CB-80 sample program for "Application CP/M"  ¥
                              行の「¥」あるいは「＼」のあとは無視される.

        INTEGER I,H,A ……FORTRAN風の整数宣言.

                 ¥input message out ……このようにコメントは自由に書く.
        PRINT
        PRINT    "1+2+3+....n = x"
INP.MSGOUT:………このようにラベルを使用できる. ラベルにピリオドを含めてもよい.
        INPUT    "input n (n = 1...250)     -->" ; INPVAL
        IF       (INPVAL<1) OR (INPVAL>250)   }                ¥
                     THEN    GOTO INP.MSGOUT  }
                                              「¥」あるいは「＼」により，同
                 ¥compute                      一行に書かなければならない
        GOSUB    CALC.SUB ………ラベルでサブルーチンを   ものでも，このように複数行
                           呼んでいる.          に分けて書くことができる.
                 ¥out result
        PRINT
        PRINT    "1+2+3+...." ; INPVAL ; "= " ; ANSX

                 STOP     ¥program run end ……プログラムの実行の終わり.


                 ¥calc subroutine
CALC.SUB:………ラベルによるサブルーチン.
        P=       INPVAL+1
        ANSX=    P * 0.5 * INPVAL
                 RETURN
                              ラインNo.がまったくないのと，自由な書式に注目.
¥List end.
```

Figure-0.1.1　コンパイラ形BASIC CB-80のソース・ファイル
（1+2+3+…+nを求めるプログラム.「応用CP/M」より）

このようにBASIC言語のひとつをとっても，いろいろな種類があり，これらはPC-8001などの一般に普及しているパーソナル・コンピュータ上でも実行できるのです．PC-8001はけっしてN-BASICだけの専用コンピュータではありません．またパソピアはけっしてT-BASICだけの専用機ではないのです．

さらに言えばパーソナル・コンピュータはBASIC言語専用のコンピュータでもありません．本書でその基礎を学ぶアセンブラを始め，COBOL（コボル），FORTRAN（フォートラン），PASCAL（パスカル），C（シー），……など各種の言語を使うこともできます．Z-80や8080，8085CPUのパーソナル・コンピュータでどこまで可能なのか．興味のある方は私の著になるCP／M3部作の「応用CP／M」（アスキー）にどこかで目を通されるとよいと思います．11種類の言語，各種アセンブラなどの実例が，具体的に示されています．そしてこれらのアセンブラや各種コンパイラ言語で開発された，マシン語による応用プログラムやビジネス・プログラムなどが，多くの人たちのパーソナル・コンピュータ上で自由に利用できるのです．

さて，パーソナル・コンピュータは，ただのBASICマシンではなく，コンピュータであることの意味が理解できたでしょうか．だからこそ本格的にコンピュータを応用しようと考えているユーザーは，パーソナル・コンピュータに組込みのBASIC言語を離れて，外のはるかに広いコンピュータの世界を知らなければなりません．

「コンピュータを知る」つまり「コンピュータ・アーキテクチャを知る」ことこそ最も重要なことでありながら，多くのBASICユーザーに適確に理解されているとは言い難いのです．

BASIC言語では大きなプログラミングができない．たとえ強引に作り上げたとしてもプログラムの流れが非常にわかりにくくなる．プログラムの開発環境が整っていない．コンパイラ形BASICではないマシン組込みのBASICでは，実行速度が非常に遅い．などのBASIC言語の限界についてここで詳しく説明するまでもな

いでしょう．もしあなたが遊びではなく，まじめにコンピュータに取り組もうとしているならば，このことはすでに気付いているかも知れません．

本書を最初から順にしっかりと読んでいってください．BASICから離れ，コンピュータを知るために必要な基礎知識をまず身に付けることが大切なのです．

マシン語を学ぼう

BASICには限界があることは，さきほどお話ししたとおりです．では，コンピュータを理解するにはどうすればいいのでしょうか．方法はいろいろあると思いますが，まずマシン語の学習からCPUの働きを知り，そこからコンピュータ全体の考え方を身に付け，その後アセンブラや他の言語に本格的に取り組むのが最も効果的な学び方でしょう．

幸いなことに，現在，多くのパーソナル・コンピュータは，その心臓部にZ-80という8ビットのCPUを使っています．このZ-80のマシン語をプログラムするための「ニーモニック」はたいへん使いやすい形式にまとめられており，また8ビットのCPUは，コンピュータの基本を学ぶには最も適したCPUなのです．だからZ-80や，その同系の8080，8085などのCPUを使ったパーソナル・コンピュータのユーザーは，マシン語を学んで，そこからコンピュータの基礎を理解していくのにたいへん有利であると言えます．

マシン語を学んでいくと，CPUの働きが理解できるようになります．CPUの働きが理解できるようになれば，あとはどんどんコンピュータというものがわかってくるでしょう．

この段階になれば「コンピュータのアーキテクチャ」を一応理解したと言えます．コンピュータ・アーキテクチャを一応身に付けたあとは，おそらくCP／MなどのOS（オーエス）（オペレーティング・システム）を利用し，その上で本格的なアセンブラを使ってマシン語のソフトを開発したり，PASCAL（パスカル），C（シー），FORTH（フォース），PL／M（ピーエルエム），COBOL（コボル）等の開発用言語や実務用言語を使って，ハイレベルなソフトを開発

することになるでしょう．そしてさらにそのさきにもコンピュータ・スペシャリストとしての道は果てしなく続いているのです．

本書からまず，マシンの基礎を学んでください．本書に書かれている基礎知識が身に付いたとき，あなたの前にコンピュータの世界が大きく開けていることに気付くはずです．

1

マシン語を学ぶ道具
MCモニタ

●

コンピュータに「メモリ」があることはすでにみなさんもご承知のとおりです．そのメモリに書き込まれているデータを見たり，メモリに任意のデータを書き込んだり，メモリに書き込まれているマシン語のプログラムを実行したりするなど，マシン語レベルでコンピュータの基本的な操作を行なう機能を持ったプログラムを「モニタ・プログラム」（あるいは単に「モニタ」）と呼んでいます．みなさんのパーソナル・コンピュータには，この「モニタ」が付属している機種も，いない機種もあるでしょう．パーソナル・コンピュータに付属しているモニタの機能は，これらの最も基本的な機能の他に，いくつかの機能が追加されています．しかし，マシン語の基礎を学ぶには，以下に挙げる３つの機能を持つモニタがあれば十分でしょう．

1．メモリの内容を見る（メモリに記憶されているデータを見る）．
2．メモリにデータを書き込む．
3．メモリの任意のアドレス（番地）からプログラムを実行する．

本書は，この３つの機能を持った独自のモニタを作成し，それを学習道具として使い，みなさんがマシン語の基礎を理解できるよう解説していきます．では本章で，このモニタを作成し，その使い方を説明しましょう．

1.1

●MCモニタの作成

本書では各パーソナル・コンピュータに共通の独自のモニタを作成して，それをマシン語を理解するための道具(ソフトウェアの世界でも「ツール」つまり道具と呼びます)としてフルに活用します．

この独自のモニタは，さきの3つの機能のみを持った実に簡潔なもので，BASICで書かれています．みなさんのパーソナル・コンピュータに，それぞれの機種独自のモニタが付属していても，本書では使用しませんので，このモニタ・プログラムを作成してそれを使ってください．しかし，ある程度，本書の理解が進めば，機種付属のモニタを使って同じように試みることも良い勉強になると思いますので各自で試みてください．

本書のモニタは，みなさんが将来，8ビットのパーソナル・コンピュータを本格的に使用するようになった場合，かならず使うことになるであろうCP／M*というOS**(オペレーティング・システム)に含まれているDDT(ディーディーティー)というマシン語のデバッグのためのプログラムと，コマンドの使い方をほぼ同じにしてあります．DDTはもちろんもっと多くの高度な機能を備えていますが，本書のモニタの3つの機能については，CRTへの表示も含めてほぼ同じに作ってあります．なお，このDDTのコマンド形式は，他の多くのモニタや，デバッガ類に影響を与えており，たとえばNECのPC-8801に付属のモニタも，このDDTと非常によく似たコマンド形式を採用しています．

さて，前置きはこのぐらいにして，本書独自のモニタの作成にはいりましょう．

*CP／Mは8080，Z-80系における代表的なOSで，8086，68000系にも用意されている．ディスクベースで多くのソフトウェアを走らせることが可能．なお，CP／M®はDigital Research社の登録商標．
**ソフトウェア，ハードウェアなどを管理するプログラムのこと．

このモニタのプログラム名を，「MACHINE CODE MONITOR」としましょう．略して「MCモニタ」とでも呼ぶことにします．

MCモニタの3つの機能は，次の3つのコマンドによって実行します．

Dコマンド	メモリの内容を見る(Dump)コマンド．
Sコマンド	メモリにデータを書き込む(Substitute：置き換え)コマンド．
Gコマンド	任意のアドレス(番地)からプログラムを実行させる(Go)コマンド．

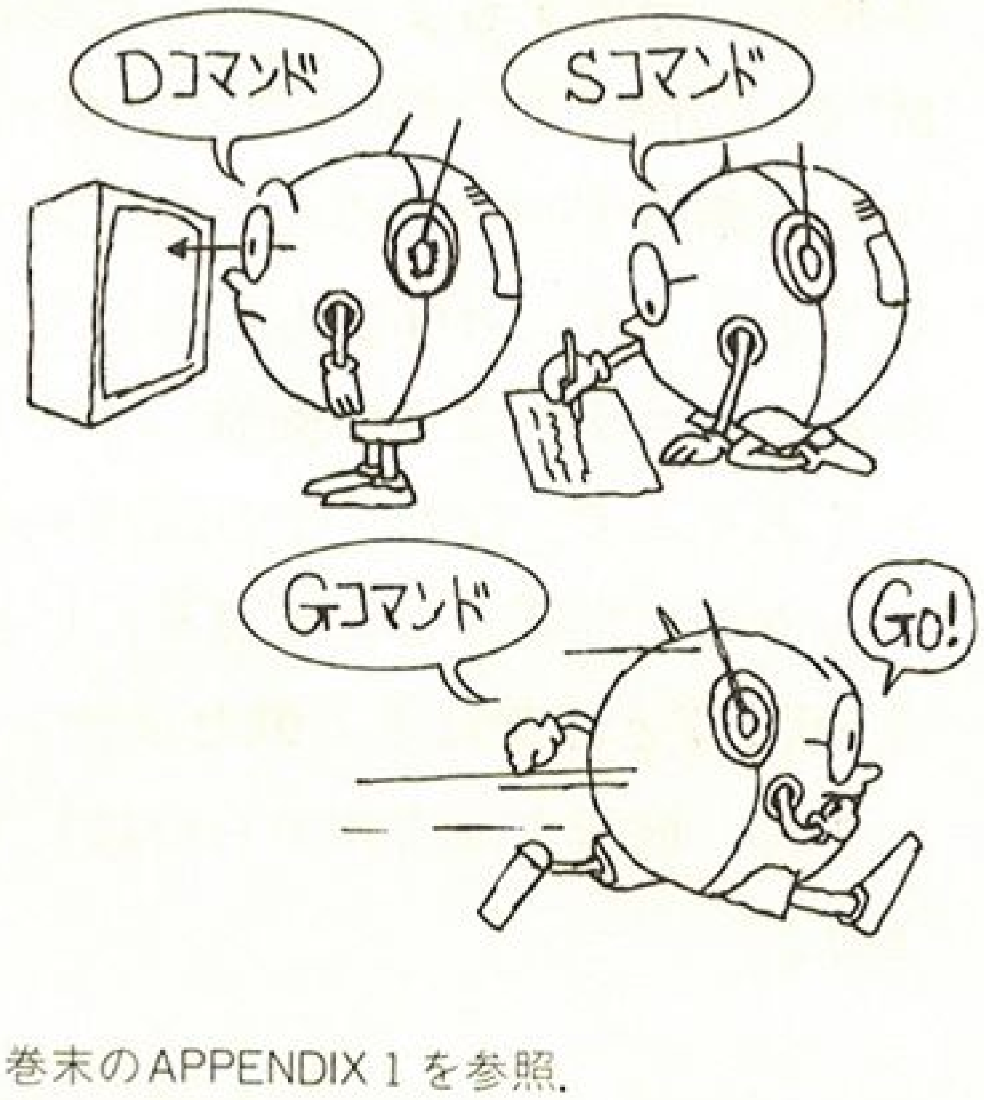
みなさんは今後このD，S，Gの3つのコマンドを使って，マシン語を学んでいくことになるわけですが，この3つのコマンドは，コンピュータの原点とも言うべき役割りを持っています．本書を読み終える頃，このたった3つのコマンドに，いかに大きな意味があるかがわかるでしょう．

MCモニタのプログラムはこのあとで示しますが，MCモニタ自身は，各パーソナル・コンピュータでの共通使用ができるように，BASIC言語でプログラムしてあります．

しかし，本書の本筋でBASICが登場するのは，このMCモニタ自身のみで，ほかにはBASICを使う個所はほとんどありません．MCモニタを打ち込み，カセットテープや，ディスクにセーブしたならば，BASICのことは忘れてください．マシン語や，コンピュータの基礎を理解するのにBASICの先入観は，かえってじゃまになります．

本書で使用するモニタ(これを「MCモニタ」と呼ぶことにしました)のプログラムリストを示します．

まずこのMCモニタのプログラムを正しく打ち込んで，カセットテープや，ディスクにセーブしてください*．



*機種によりプログラムの一部を変更する必要がある．巻末のAPPENDIX 1を参照．

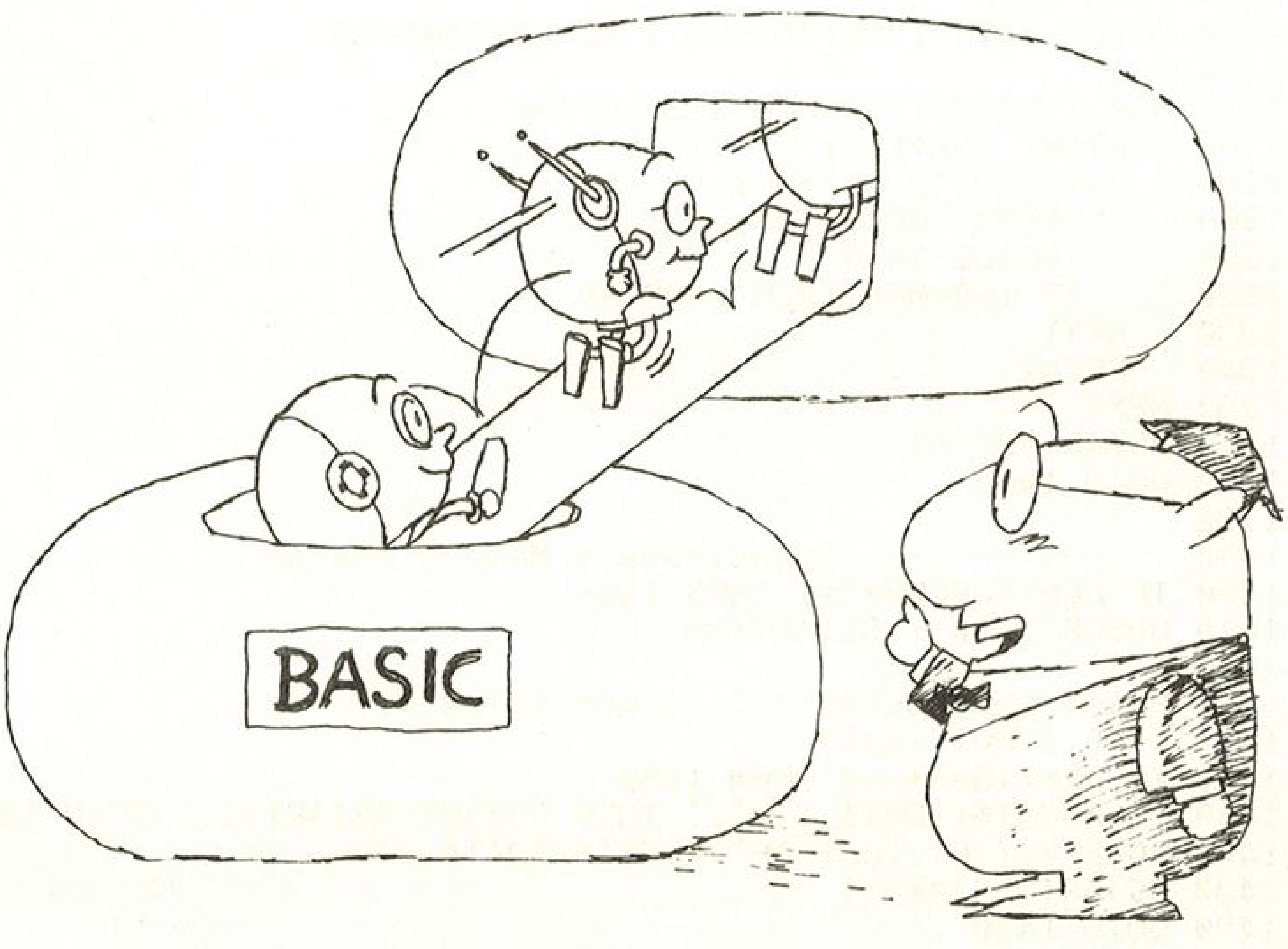
```
1000 CLEAR 300,&H8FFF ………機種ごとに変わるので注意.
1010 OPT=2 ………Dコマンドを実行する際，これが1の場合1行に8バイト表示し，2の場合16バイト表示する.
1020 WIDTH 80,25 : CONSOLE 0,25 : PRINT CHR$(12);
1030 PRINT TAB(18*OPT-9) ; "MACHINE CODE MONITOR"
1040 DEF FNA$(X)=RIGHT$("000"+HEX$(X),4)
1050 DEF FND$(X)=RIGHT$("0"+HEX$(PEEK(X)),2)
1060 '
1070 PRINT
1080 LINE INPUT "-";BUFFER$………コマンドを入力する．大文字でも小文字でも可.
1090 COMMND$=LEFT$(BUFFER$,1)
1100 IF COMMND$="D" OR COMMND$="d" THEN 1140 ┐ コマンド名に応じてそれ
1110 IF COMMND$="S" OR COMMND$="s" THEN 1380 ├ ぞれの処理ルーチンへと
1120 IF COMMND$="G" OR COMMND$="g" THEN 1500 ┘ ぶ.
1130 PRINT " What ?" : GOTO 1070
1140 '                                         ┌ MSTART：表示を開始するアドレス.
1150 '  ----------  Dump : Memory dump         └ MEN：表示を終了するアドレス.
1160 IF LEN(BUFFER$)=1 THEN MEN=MSTART+63 : GOTO 1260
1170 J=INSTR(BUFFER$," ") : IF J=0 THEN 1250
1180 IF J-4>2 THEN K=J-4 ELSE K=2 ………K：MSTARTとして何文字目からとるか.
1190 L=LEN(BUFFER$)-J : IF L>4 THEN L=4……L：MENとして何文字とるか.
1200 MSTART=VAL("&h"+MID$(BUFFER$,K,J-K))
1210 IF MSTART<0 THEN MSTART=MSTART+65536!……符号なし2進数に変換する．こ
1220 MEN=VAL("&h"+RIGHT$(BUFFER$,L))           うしないとアドレス0H～FFFFH
1230 IF MEN<0 THEN MEN=MEN+65536!              間の連続ダンプができない.
1240 GOTO 1260
1250 GOSUB 1570 : MSTART=MM : MEN=MSTART+63
1260 '
1270 FOR I=MSTART TO MEN STEP OPT*8
1280   PRINT FNA$(I);"  ";
1290   FOR Q=0 TO OPT*8-1
1300     PRINT FND$(I+Q);" ";
1310     'GOSUB 1620……アポストロフィを除くと，16進ダンプと共にアスキーダンプも行なう.
1320     IF I+Q=MEN THEN Q=OPT*8
1330   NEXT
1340   PRINT
1350 NEXT
1360 MSTART=MEN+1……パラメータの入力がなくDコマンドを実行した場合のためにMSTART
1370 GOTO 1060        を設定しておく.
1380 '
1390 '  ----------  Substitute : Memory change
1400 IF LEN(BUFFER$)=1 THEN 1420
1410 GOSUB 1570 : SSTART=MM………SSTART：書き換えるアドレス.
1420 '
1430 PRINT FNA$(SSTART);" ";FND$(SSTART);" ";
1440 LINE INPUT IDAT$
1450 IF LEN(IDAT$)=0 THEN 1480
1460 IF RIGHT$(IDAT$,1)="." THEN SSTART=SSTART+1 : GOTO 1060
1470 POKE SSTART,VAL("&h"+RIGHT$(IDAT$,2))……書き換えるデータとして3
1480 SSTART=SSTART+1                            文字以上指定した場合は右
1490 GOTO 1420                                  の2文字が有効になる.
```

```
1500 '
1510 ' ----------  Go : Execute user program
1520 IF LEN(BUFFER$)=1 THEN PRINT " What ?" : GOTO 1060
1530 GOSUB 1570
1540 IF MM>32767 THEN DEF USR0=MM-65536! ELSE DEF USR0=MM
1550 DUMMY=USR0(0)
1560 GOTO 1060
1570 '……パラメータがひとつのとき，そのアドレスをMMへとりだすサブルーチン.
1580 M=LEN(BUFFER$)-1 : IF M>4 THEN M=4
1590 MM=VAL("&h"+RIGHT$(BUFFER$,M))
1600 IF MM<0 THEN MM=MM+65536!
1610 RETURN
1620 '……アスキーダンプをするためのサブルーチン.
1630 CX=POS(0)……カーソルの横位置を保存しておく.
1640 LOCATE 7+3*8*OPT+Q,CSRLIN
1650 ADAT=PEEK(I+Q)
1660 IF ADAT<&H20 THEN PRINT "."; ELSE PRINT CHR$(ADAT);
1670 LOCATE CX,CSRLIN……カーソルをもとの位置に戻す.
1680 RETURN
```

……Gコマンドにパラメータがない場合，エラーメッセージを表示する.（1520行）

……コントロールコードのかわりにピリオドを表示する.（1660行）

Figure-1.1.1　本書で使用するモニタ（MCモニタ）のプログラムリスト



1.2

●MCモニタの使い方

作成したMCモニタの動作チェックを兼ねて，その使い方の練習を軽くやっておきましょう(その前にかならずMCモニタのプログラムをカセットテープやディスクにセーブすることを忘れないように！動作チェックをする前にかならずセーブを！).

コマンド名	機　能	コマンド形式とその機能
D (ダンプ)	メモリの内容(記憶されているデータ）を表示する．	Dxxxx␣yyyy↵* …xxxx番地からyyyy番地の間のメモリの内容を表示する． Dxxxx↵ ……………xxxx番地から64番地分(64バイト**分，xxxx～xxxx＋63)のメモリの内容を表示する． D↵ ……………………以前実行したDコマンドの最終番地の次から64バイト分のメモリの内容を表示する．
S (サブスティテュート)	メモリの内容を変更する．	Sxxxx↵ ……………xxxx番地から１バイトずつメモリの内容が表示され，必要があれば任意のデータに書き換えていく．コマンドモードに戻るときは，ピリオドのみを入力する． S↵ ……………………以前実行したSコマンドの次の番地からメモリの内容を書き換えていく．
G (ゴー)	プログラムを実行する．	Gxxxx↵ ……………xxxx番地からメモリ上のプログラム(マシン語の)を実行する．
備考：コマンドは大文字でも小文字でも，どちらで与えてもかまいません．メモリの番地指定XXXXは４桁以下でもかまいません．もし５桁以上を入力した場合は右の４桁のみが有効です．なお，入力する部分はアンダーラインで示してあります．		

Figure-1.2.1　MCモニタのコマンド/機能一覧表

＊␣はスペース（空白），↵はリターンキーを入力する意味．
**データを扱うときの単位．詳しくは２．１章を参照．

ここではMCモニタの使い方をひと通り実習しますが，その内容の意味については，次章以後で一つひとつ詳しく解説します．よって，ここでは例題通りの実行結果が得られれば，それでOKとします．その意味など今はわからなくても結構です．また，わからない用語なども気にせず進んでください．これらはあとの章で解説してありますので安心してください．

MCモニタのコマンドとその機能は，一覧表にまとめて前頁に示しました．

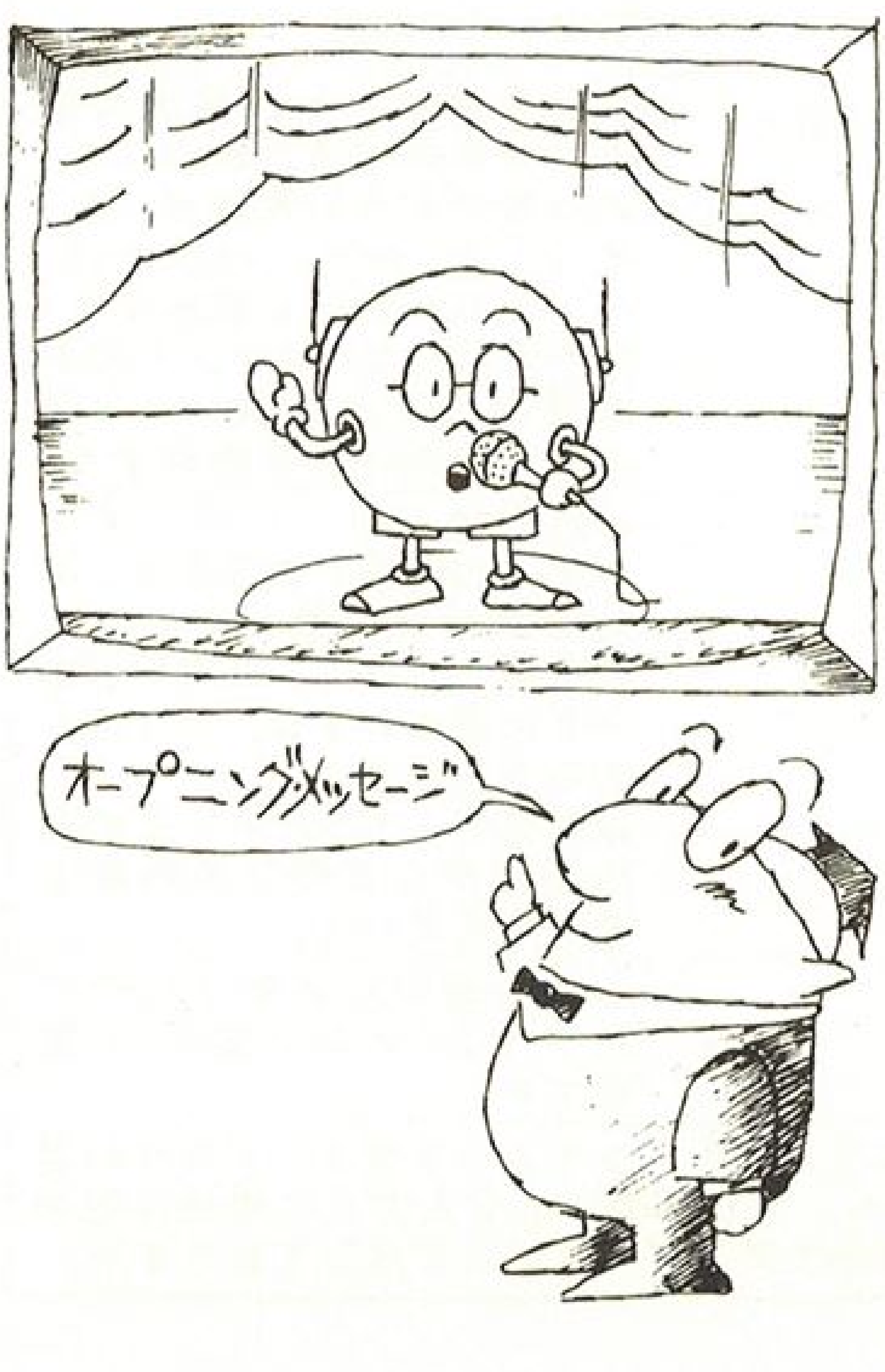
では，作成したMCモニタのプログラムをロードして，RUNしてください．

```
MACHINE CODE MONITOR
-
```

MACHINE CODE MONITOR……オープニング・メッセージ

-……プロンプト．コマンドの入力を受けつけますという表示．

Figure-1.2.2 MCモニタのオープニング・メッセージとプロンプト



このように，MCモニタの**オープニング・メッセージ***と，コマンドを受け付ける準備ができていることを知らせる**プロンプト****が表示されます．

さて，ここからはマシン語の世界です．マシン語の世界ではBASICのコマンドやステートメントは，何ひとつ意味がありませんのでそのつもりで頭を切り換えてください．

まず最初はDコマンドから，動作チェックを兼ねてコマンドの使い方を実習しましょう．

*ひとつのプログラムが起動したときに，そのプログラムが何であるかを示す表示文．

**コマンドの入力を促する表示（Prompt）．

●● Dコマンド(メモリダンプ) ●

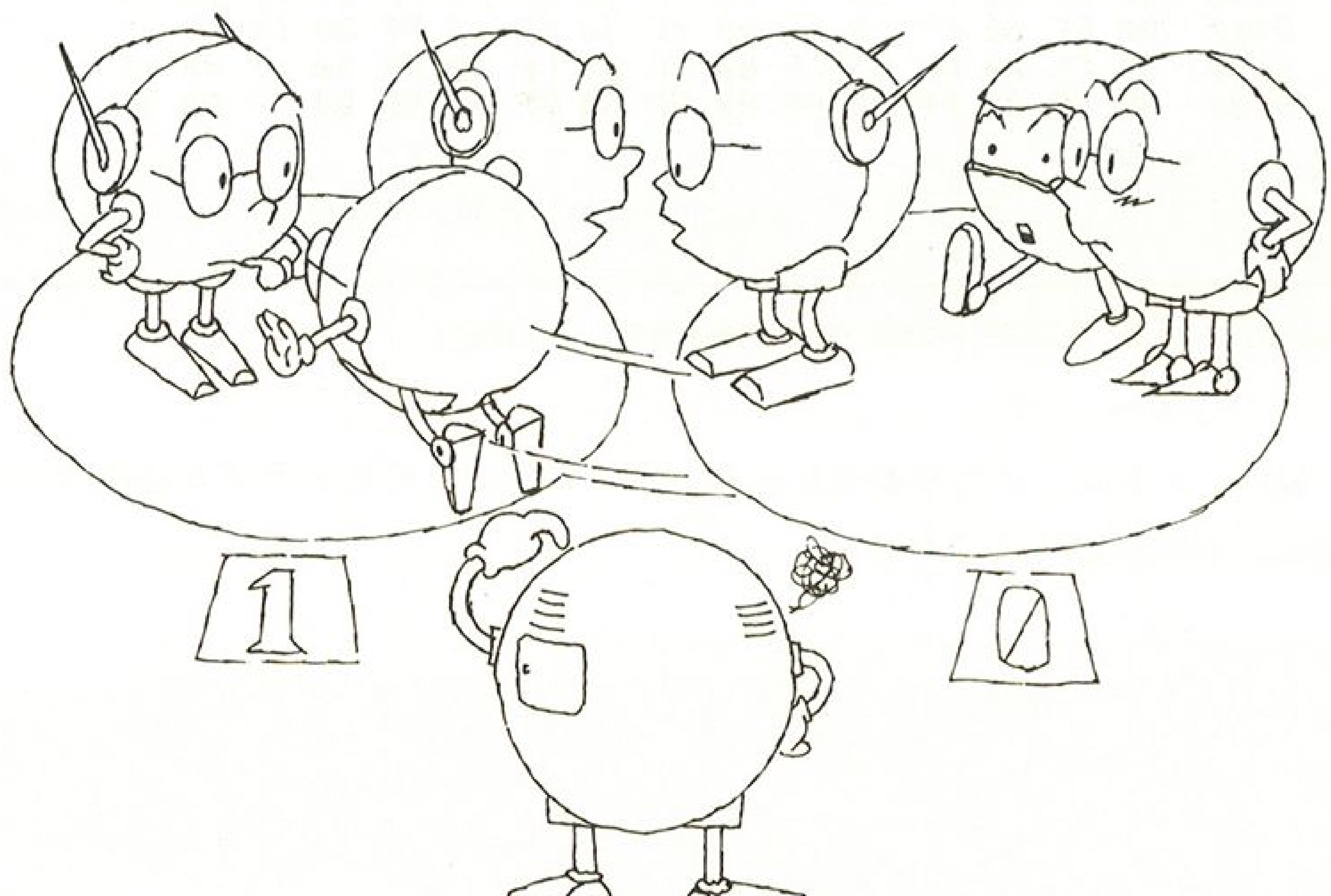
Dコマンドは，メモリの内容(メモリに記憶されているデータ)を見る(ダンプする)コマンドです．見たいと思うメモリの範囲を**アドレス***(番地)で指定することにより，その範囲のメモリの内容がダンプされます．

Dコマンドの使い方には次に示す3通りがあります．

<u>Dxxxx␣yyyy↵</u>……	xxxx番地からyyyy番地までをダンプする．
<u>Dxxxx↵</u>……………	xxxx番地から64番地分をダンプする．
<u>D↵</u>……………………	以前に実行したDコマンドの最終番地の次から64番地分をダンプする．
(下線部分を入力)	

以上の3通りの使い方があります．ここでxxxxやyyyyは最大4桁の16進数での値ですが，たとえば「0010」は「10」と書いてもOKです．

とにかく実際に実行してみましょう．実行例を見れば一目瞭然だと思います．



*メモリに付けられた番地のこと．詳しくは2．1章を参照．

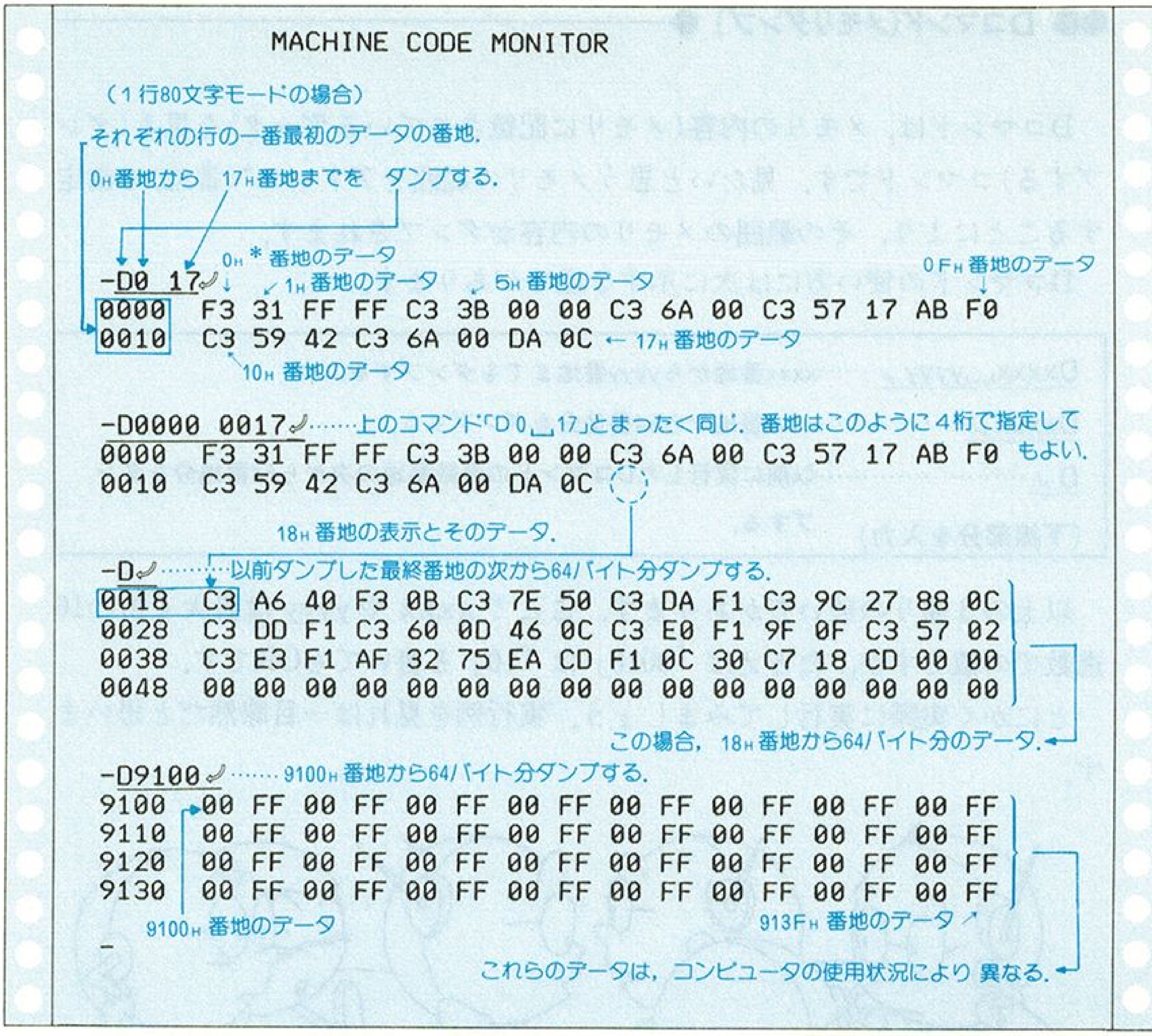


Figure-1.2.3 Dコマンドの使い方(1行80文字モードの場合)

Dコマンドは，どこの番地をどのようにダンプしてもトラブルは発生しません．自由に試みてください．

＊アドレスはふつう16進数で表示する．16進数は10進数と区別するため，終わりに「H」を付ける．

(1行40文字モードの場合)

```
-D0 17↵
0000  F3 31 FF FF C3 3B 00 00
0008  C3 6A 00 C3 57 17 AB F0
0010  C3 59 42 C3 6A 00 DA 0C

-D0000 0017↵
0000  F3 31 FF FF C3 3B 00 00
0008  C3 6A 00 C3 57 17 AB F0
0010  C3 59 42 C3 6A 00 DA 0C

-D↵
0018  C3 A6 40 F3 0B C3 7E 50
0020  C3 DA F1 C3 9C 27 88 0C
0028  C3 DD F1 C3 60 0D 46 0C
0030  C3 E0 F1 9F 0F C3 57 02
0038  C3 E3 F1 AF 32 75 EA CD
0040  F1 0C 30 C7 18 CD 00 00
0048  00 00 00 00 00 00 00 00
0050  00 00 00 00 00 00 00 00

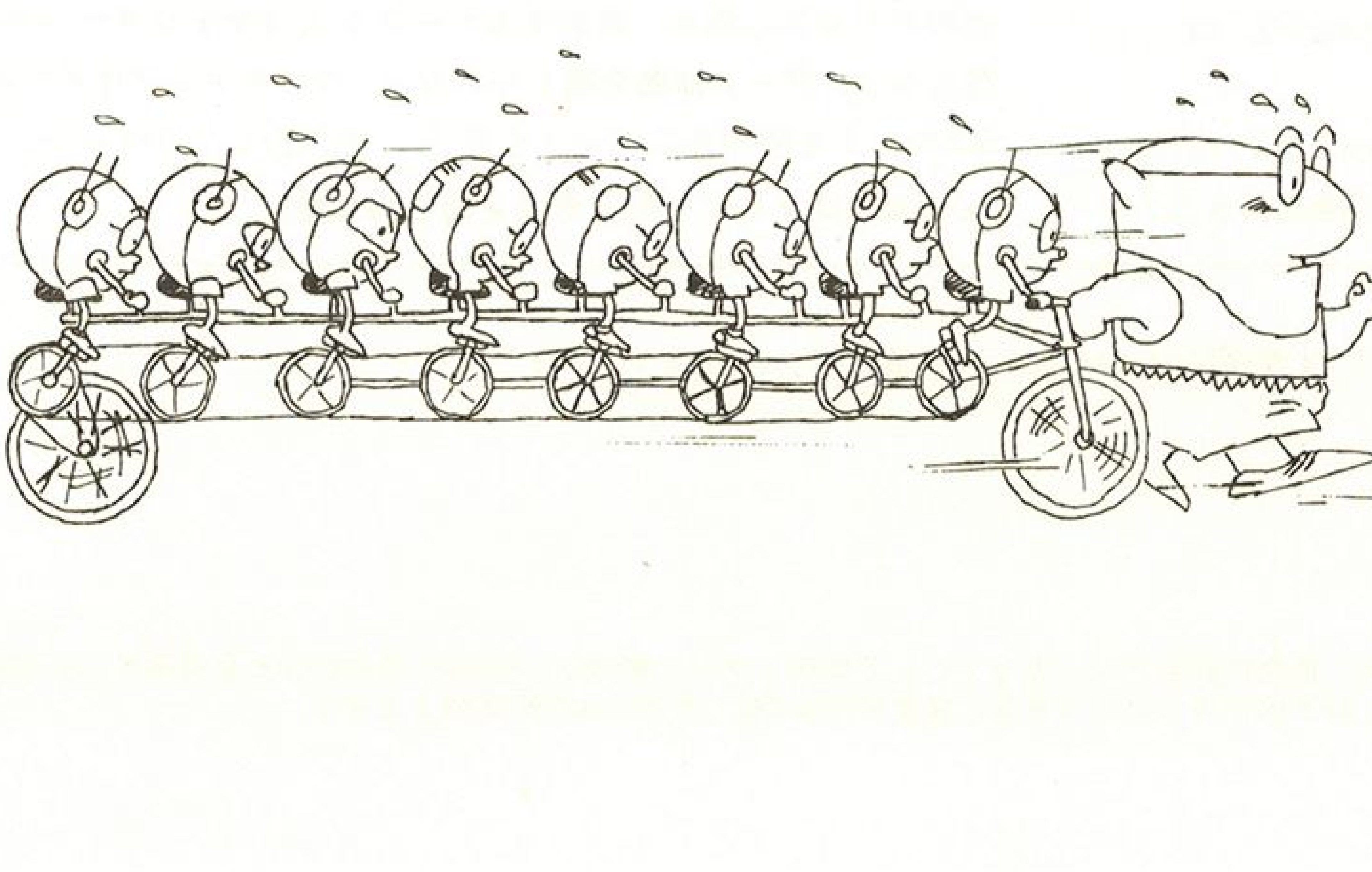
-D9100↵
9100  00 FF 00 FF 00 FF 00 FF
9108  00 FF 00 FF 00 FF 00 FF
9110  00 FF 00 FF 00 FF 00 FF
9118  00 FF 00 FF 00 FF 00 FF
9120  00 FF 00 FF 00 FF 00 FF
9128  00 FF 00 FF 00 FF 00 FF
9130  00 FF 00 FF 00 FF 00 FF
9138  00 FF 00 FF 00 FF 00 FF

-
```

MCモニタのプログラムのラインNo.1010の「OPT＝2」を「OPT＝1」に変更すると，Dコマンドによる表示が1行に8バイトとなる.

(Figure-1.2.3と同じ実行例)
左端の番地表示も8番地おきの表示となることに注目.

Figure-1.2.4 Dコマンドの使い方(1行40文字モードの場合)



●● Sコマンド(メモリ内容の変更) ●

Sコマンドは，指定した番地のメモリの内容を1バイトずつ表示し，必要があれば任意のデータに変更していくコマンドです．よって，もしメモリの内容を変更しなければ，1番地ごとのダンプ・コマンドとしても使えます．

Sコマンドでメモリの内容を変更する場合は，変更するメモリの番地に注意が必要です．何しろメモリの内容が変ってしまうのですから，その番地がたとえば，当MCモニタのプログラムが格納されているエリアだったり，そのプログラムがRUNするために必要なワーキングエリアだったりしたらたいへんです．プログラムが壊れてMCモニタは働かなくなるでしょう．巻末のAPPENDIX 1で確認してください．

またRAMエリアではなくROMエリアのメモリ内容をSコマンドで変更しても，実際には書き換わっていないことは当然です．

Sコマンドは次のような形式で使います．

<u>Sxxxx ↵</u>		………………	アドレスxxxxからメモリ内容を変更するモードにはいる．
xxxx	aa	<u>a'a' ↵</u>	アドレス値xxxxとその内容「aa」が表示されるので，変更しようとする値「a'a'」を入力しリターンすると，aa→a'a'に書き換えられる．
xxxx+1	bb	<u>b'b' ↵</u>	自動的に次のアドレスとその内容が表示される．同様にbb→b'b'に書き換えられる．
xxxx+2	cc	<u>↵</u> ……	次のアドレスに移る．変更するデータを入力せずリターンのみ行なうと，データは書き換えられない．つまりダンプするだけ．
xxxx+3	dd	<u>. ↵</u> …	Sコマンドを終わるにはピリオド「.」を入力してリターン．
(下線部分を入力)			(xxxxはDコマンドと同じ最大4桁の16進数)

では実際にSコマンドを実行してみましょう*．

*9000_H 番地付近をフリーエリアとして使用できない機種は，$D000_H$ 番地台などを使用する必要がある．以下のリスト中の9をDに変更すればOK．巻末のAPPENDIX 1を参照．

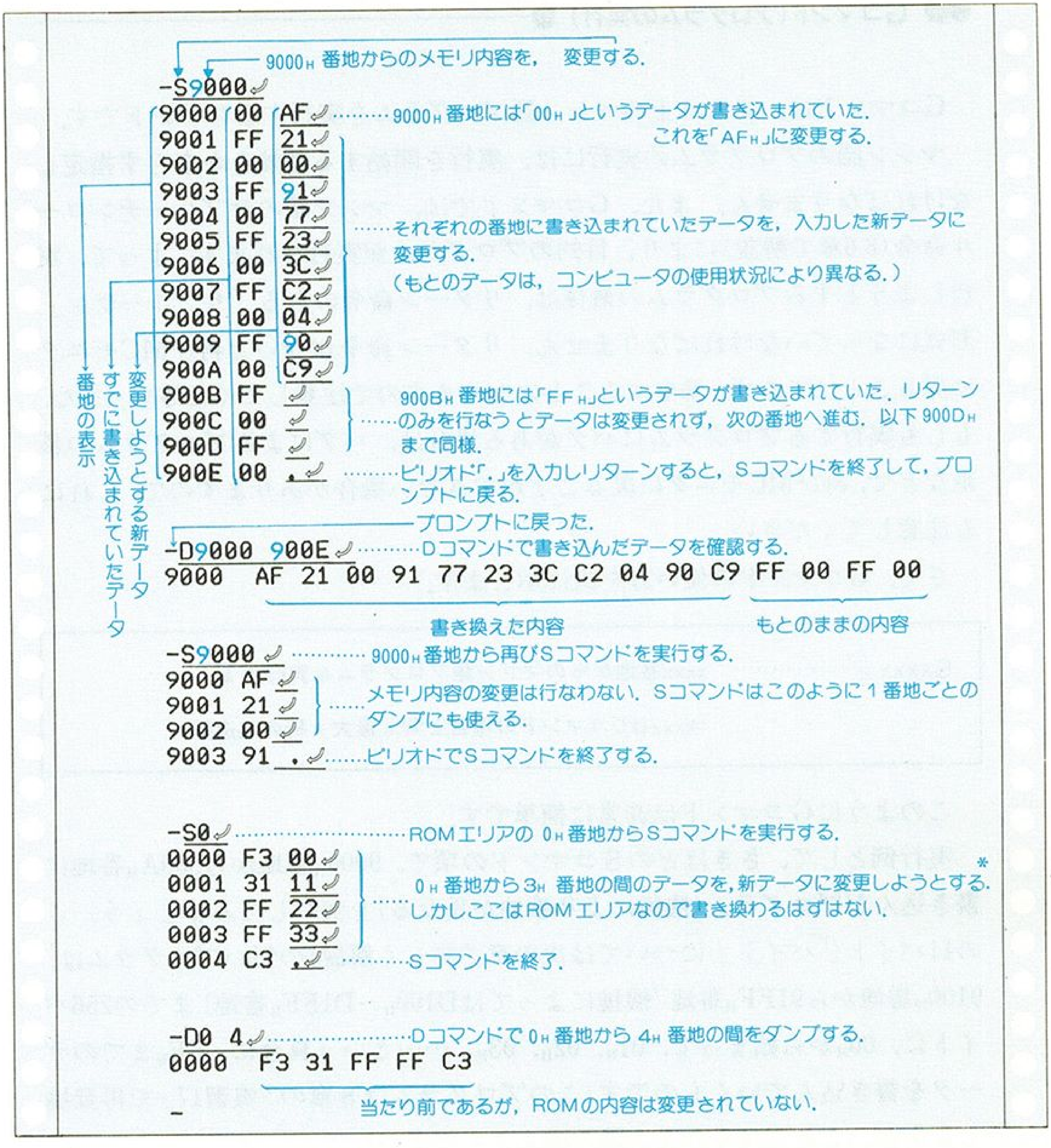


Figure-1.2.5 Sコマンドの使い方

この実行例で変更したメモリ内容(9000_H〜$900A_H$番地)**は, あるひとつのマシン語のプログラムになっています. これは次のGコマンドで実行します.

*0_H 番地付近のエリアがROMでない機種の場合は, この書き換えは行なわないこと.

**機種によっては$D000_H$〜$D00A_H$番地となる. なおアドレスはすべて16進数であるため, 10進数と区別するために最後に「H」を付ける.

●● Gコマンド(プログラムの実行) ●

Gコマンドは，メモリ上のマシン語プログラムを実行するコマンドです．

マシン語のプログラムの実行には，実行を開始する番地をかならず指定しなければなりません．また，Gコマンドでは，マシン語のサブルーチンコール命令(8.6章で解説)により，目的のプログラムが実行されます．よって，実行しようとするプログラムの最後は，リターン命令のある「サブルーチン」形式になっていなければなりません．リターン命令がないと再びMCモニタに戻ることはできず，暴走することになりますので注意してください．また，もしも実行するプログラムにバグがある場合は，バグによるプログラムの暴走などで，再びMCモニタに戻ることができない場合がありますので，これにも注意してください．

さて，Gコマンドの使い方を次に示します．

Gxxxx ↲ ……………	xxxx番地からのマシン語プログラムを実行する． (xxxxはDコマンドの場合と同じ最大4桁の16進数)

このようにGコマンドは非常に簡単です．

実行例として，さきほどのSコマンドの項で，9000_H番地から$900A_H$番地に書き込んだプログラム(機種により番地が異なる)を実行してみましょう．この11バイト(「バイト」については次の章で詳しく解説)の短いプログラムは，9100_H番地から$91FF_H$番地(機種によっては$D100_H$～$D1FF_H$番地)までの256バイトに，00_Hから始まって，01_H，02_H，03_H，……という具合に，FF_Hまでのデータを書き込んでいくものです(このプログラムは8章の「実習17」で再登場しますが，ここではそのまま読み進んでください)．

では実行例を示しましょう．

```
  ┌─9000H番地から* プログラムを実行する
-G9000↵

- ………プログラムの実行が終わって，再びプロンプトに戻った.
```

Figure-1.2.6　Gコマンドの使い方

9000$_H$番地（機種によってはD000$_H$番地）からのプログラムの実行は瞬時に終わり，再びMCモニタのプロンプトに戻っています．実行結果を調べてみましょう．Dコマンドで，9100$_H$番地から91FF$_H$番地の間をダンプしてみれば結果がわかります．

その実行例を示します．この例では念のため9207$_H$番地までダンプしています．

```
-D9100 9207↵ ………Dコマンドでダンプして，Gコマンドによる実行結果を確認する.*
9100  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
9110  10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
9120  20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
9130  30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
9140  40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
9150  50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
9160  60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
9170  70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
9180  80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
9190  90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
91A0  A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
91B0  B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
91C0  C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
91D0  D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF
91E0  E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
91F0  F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
9200  00 FF 00 FF 00 FF 00 FF
        ↑
     対象外の部分
-
```

00H からFFHまでのデータが，9100H 番地から順に書き込まれている.
Gコマンドにより，9000H 番地からのプログラムは正常に実行されたことがわかる.

Figure-1.2.7　Gコマンドによるプログラム実行結果の確認

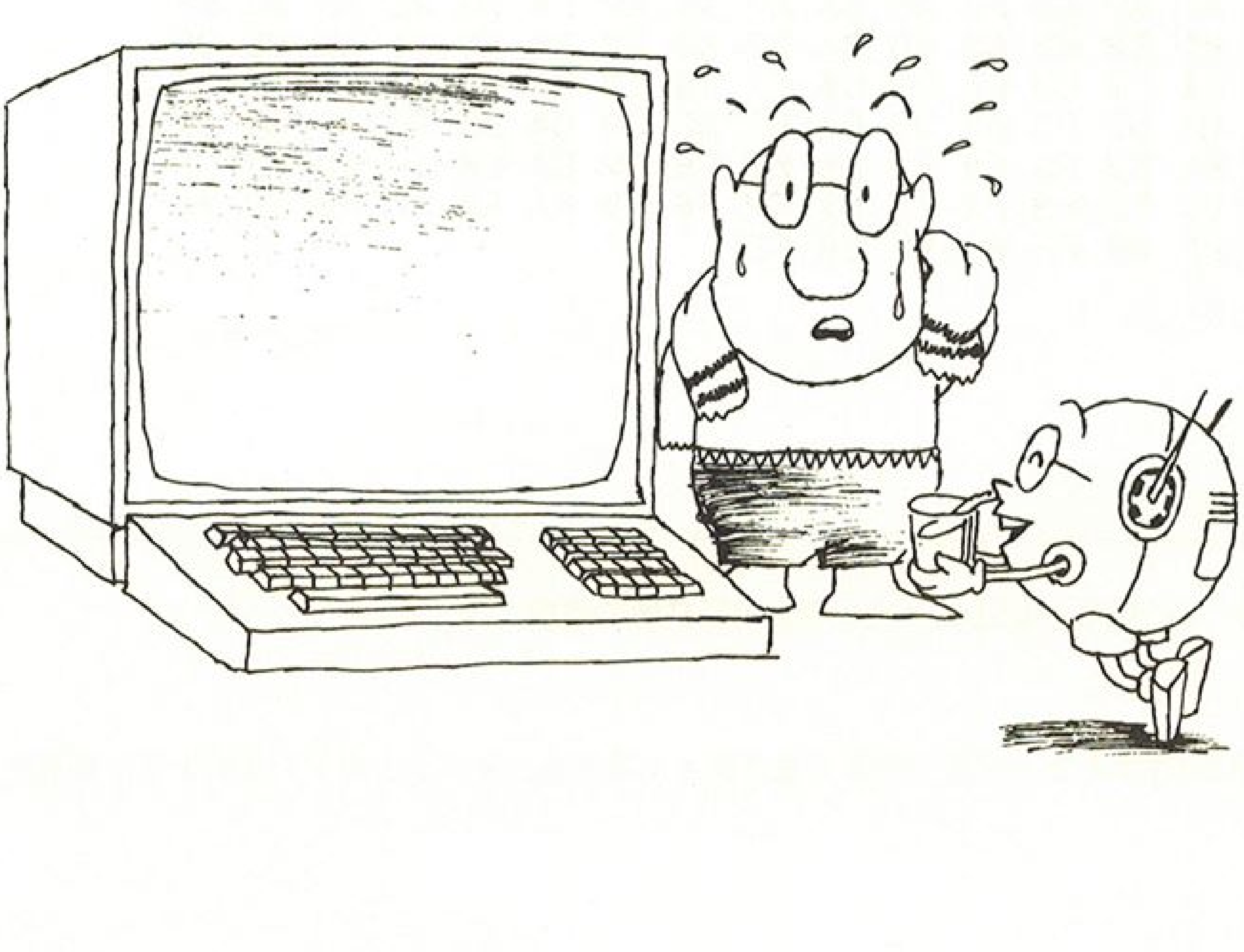
*Sコマンドで D000H 番地台からメモリ内容を書き換えた場合は，マークされている9をDに変更する必要がある．

このように00, 01, 02, 03……FE, FFと数字が並んでいればOKです．この数字は，10進数で00, 01, 02, 03……254, 255までを16進数で表わしたものです．

このような結果が得られたでしょうか．番地については各機種により異なりますが，もしうまくいかない場合は，BASICで書かれたMCモニタのプログラムに打込みミスがあるか，またはMCモニタのSコマンドでメモリに書き込んだデータにミスがあるか（機種別の使用可能なフリーエリアを間違えないように），などが考えられます．もう一度チェックしなおしてください．

D, S, Gのすべてのコマンドがうまく働けば，MCモニタは完成です．今後はこのMCモニタをマシン語学習の「tool（道具）」として，フルに活用していきます．

本章では，MCモニタの使い方を一応理解しておけば結構です．次章以後も例題にはすべて実行例を示しますので，徐々に扱いに慣れればよいでしょう．



2

メモリに関する基礎知識

「32Kバイトの ROM」，「64Kバイトの RAM」などという言葉が新聞，雑誌などで日常よく使われています．みなさんもすでに「メモリ」とはプログラムやデータを記憶するところ，というような概念を持っていると思います．本章ではこの「メモリ」について，アドレス，バイト，ビットなどのソフトウェア的考え方の基本を具体的に解説していきます．

これらの概念を理解するには，16進法と2進法の考え方を利用することが不可欠なのですが，本章ではまずメモリの具体的な「姿」を見ることにします．そのあと（3章）で16進法と2進法について，コンピュータに即した解説を行なっていますので，効果的にメモリや16進法・2進法の関係が理解できると思います．

もし16進法や2進法について早く知りたい方がいても，本章はひと通り読んでください．本章での知識がある方が，それらはずっと理解しやすくなります．そして本章をひと通り読んだあと次の章に進み，16進法や2進法がある程度理解されてきたら，再び本章に戻ってみてください．「メモリ」についてさらに理解が深まるものと思います．

2.1

●アドレス，バイト，ビット

まず，メモリに関しては，次の3つの基本的な概念を理解しなければなりません．

アドレス	何万とあるメモリの中から目的のひとつのメモリを捜し出し，リード／ライト（読出し／書込み）するために，メモリの個々に付けられている「番地」のこと．
バイト	8ビットをひとつの単位として，それをバイトと呼ぶ．つまり1バイトは8ビット構成である．8ビットのマイクロコンピュータではメモリの基本的な単位でもあり，コンピュータがメモリをリード／ライトする場合の単位でもある．それぞれのバイトは個々のアドレス（番地）を持っている．
ビット	コンピュータが取り扱うすべてのデータの最小単位であり，1ビットのデータは「0」か「1」のいずれかの値をとる．8ビットで1バイトを構成する．

私たちは1章でマシン語を学ぶためのツール(道具)であるMCモニタを作りました．このMCモニタのDコマンド(ダンプコマンド)を使えば，任意のアドレスのメモリに記憶されているデータを見ることができます．一般的に，メモリに記憶されているデータのことを，**メモリの内容**と表現していますので本書でもこのように表現することにします．

メモリはバイト単位に区切られていて，各バイトにはその住所表示であるアドレスが付けられています．Dコマンドは，アドレス(番地)を指定することにより，指定されたメモリの内容をCRTに表示するコマンドであるわけです．

MCモニタを使って実習する前に，「アドレス」，「バイト」，「ビット」についての予備知識を解説しておきましょう．

●● アドレス(Address) ●

すべてのメモリの1バイトごとに付けられているアドレスは，0_H*番地から始まります．1番地からではありませんので注意してください．Z-80などのCPUを持つコンピュータのメモリは，アドレス0_H番地から始まり，

0，1，2，3，…………E，F，10，11，12，…………1E，1F，
20，21，22，………FE，FF，100，101，102，………FFE，FFF，
1000，1001，1002，……………………………………………FFFE，FFFF

という具合に，16進の$FFFF_H$番地までの($FFFF_H$は10進の65535)合計65536個(バイト)存在します．このことを「Z-80などのCPUのアドレス空間は，0_H番地から$FFFF_H$番地である」というような表現をします．

よく，「64Kバイト・フル実装」などと言われるのを耳にしますが，この意味は「65536バイト・フル実装」ということです．メモリに関しては，

1Kバイト＝1024バイト＝2^{10}バイト

として数えます．よって，

64×1024＝65536

となるわけです．64Kバイトと言っても64000バイトではなく，実際は65536バイトもあるわけです．

*アドレスはすべて16進数．10進数と区別するために数値の最後に「H」を付ける．ただし1桁の0～9は「H」を付けなくても同じ値．

●● バイト(Byte) ●

「バイト」はデータの基本単位です．CPUとメモリ間で行なわれるすべての処理は，このバイト単位でデータの出し入れが行なわれます．1バイトは8ビットで構成されており，メモリ上ではそれぞれのバイトには個別のアドレスが付けられています．

1バイトで表現できるデータの数値は，0_HからFF_H(10進数では0から255)の間のひとつの値です．なぜ0_HからFF_Hの値になるのかは，のちほど詳しく解説します．

●● ビット(Bit) ●

「ビット」の概念こそコンピュータ理解のキーポイントとでも言うべきものです．この概念がつかめないうちは**コンピュータ・アーキテクチャ***を理解することはできないでしょう．

ビットの世界は「1」か「0」かの世界です．コンピュータの内部は，メモリもCPUも，すべてこのビットの組合せにより動作しています．つまりコンピュータの内部のデータはすべて「1」と「0」のふたつの値だけで各種の処理が行なわれています．

ビットはコンピュータで処理されるデータを構成する最小単位であり，ビットをさらに細かく分割することはできません．

ビットは「1」か「0」の世界であるわけですが，これは言い換えれば「ある」か「ない」かと言うことです．具体的には電圧が「ある」か「ない」かであり，多くのコンピュータは，このふたつの状態を電圧の「＋5V」と「0V」で区別しています．

次に，メモリを例にとり，「アドレス」，「バイト」，「ビット」の関係を図で示しましょう．もし16進と2進についての知識が必要であれば次章を随時参照してください．

*コンピュータのソフト，ハードに渡るシステムの構造のこと．

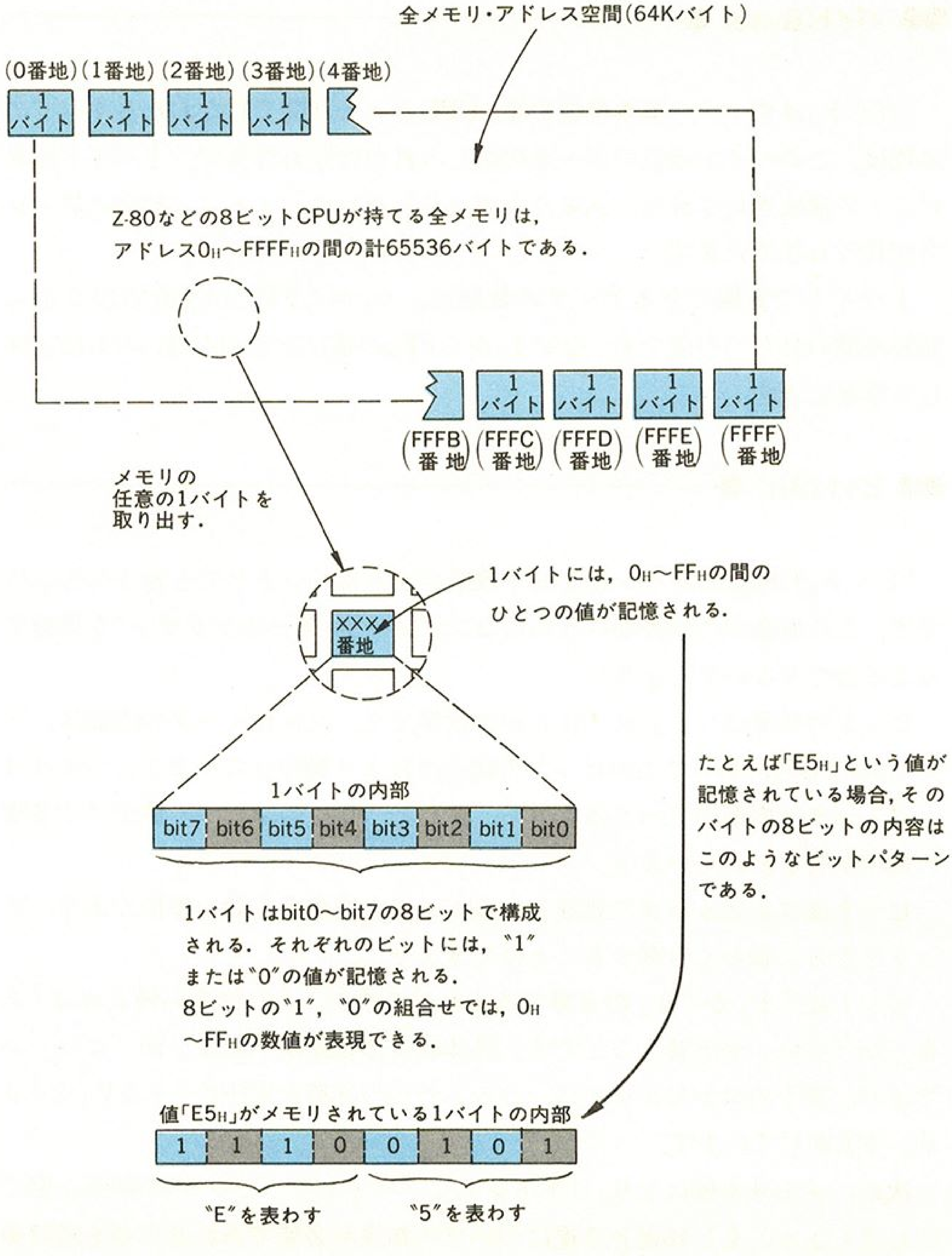


Figure-2.1.1 アドレス，バイト，ビット，「メモリ」の概念図

2.2

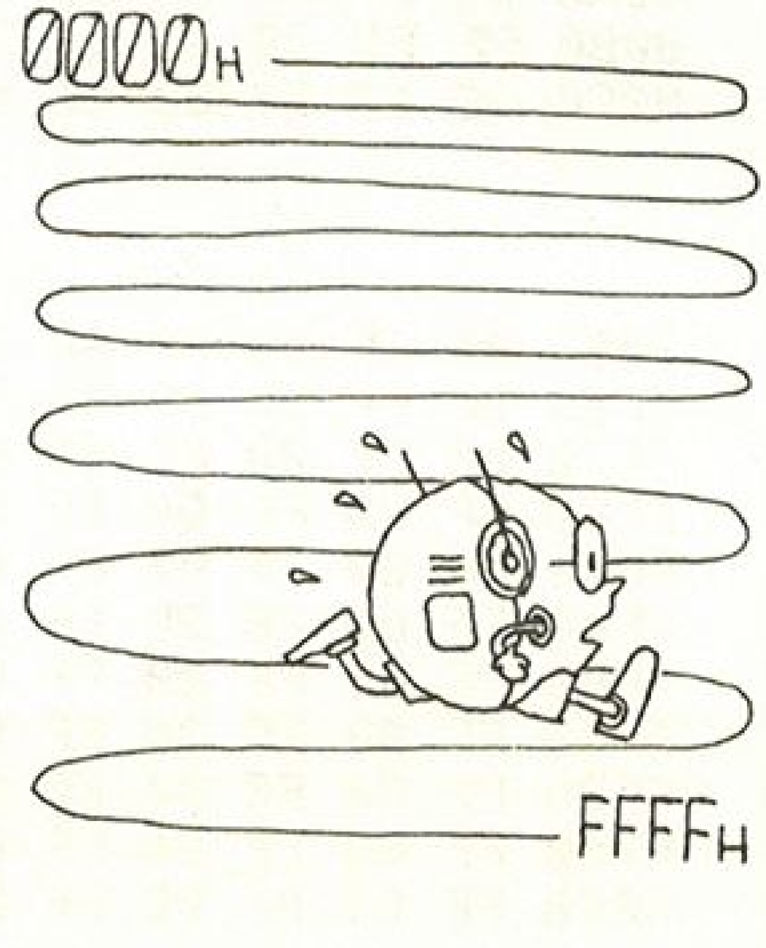
●メモリに記憶されているデータを見る

では，1章で作成した，「MCモニタ」を使って，メモリに関して具体的な解説を行なっていきましょう．

まず，Figure-2.1.1（アドレス，バイト，ビット，「メモリ」の概念図）の上部の図にイメージとして描かれている，Z-80などの8ビットCPUを使ったコンピュータのメモリの全アドレス，65536バイトをすべてダンプして，その様子を実感してみましょう．

MCモニタをロードし，RUNしてください．全アドレス(0000_H～$FFFF_H$)のメモリ内容をダンプするには，ダンプコマンド(Dコマンド)を次のように実行します．

D 0 ␣FFFF ↵



この実行例をFigure-2.2.1に示します．アドレス0000_Hから始まり延々と$FFFF_H$までの表示が続きますが，30分以上（PC-8801のN-BASICの場合）かかりますので，適当なところでBREAK*して結構です．

このように，0_H番地から$FFFF_H$番地まで全アドレスの64Kバイト(65536バイト)のメモリ内容がダンプされました．

＊MCモニタはBASICのプログラムで実行されているので，Gコマンドでマシン語のプログラムが走っているとき以外は，いつでもBREAKしてBASICに戻ることができる．

```
-D0 FFFF
0000 F3 31 FF FF C3 3B 00 00 C3 6A 00 C3 57 17 AB F0
0010 C3 59 42 C3 6A 00 DA 0C C3 A6 40 F3 0B C3 7E 50
0020 C3 DA F1 C3 9C 27 88 0C C3 DD F1 C3 60 0D 46 0C
0030 C3 E0 F1 9F 0F C3 57 02 C3 E3 F1 AF 32 75 EA CD
0040 F1 0C 30 C7 18 CD 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 F3 CD 4C 16 AF 32
0070 60 EA 3D 32 61 EA 01 81 00 C5 01 00 FF C5 C3 F6
0080 08 CD B0 F1 3A 55 EB ED 47 3E FF D3 E4 32 55 EA
0090 ED 5E FB 01 81 3C C3 10 3C C5 D5 F5 CD B3 F1 32
00A0 B9 ED 3E 01 38 02 3E 02 CD 21 01 F1 F5 47 38 0F
00B0 28 10 CD 5A 01 E6 40 CA 68 01 CD 39 01 18 03 CD
00C0 4C 01 CD 5A 01 F1 D1 C1 B7 C9 C9 DB 40 E6 08 EE
00D0 08 C9 C5 D5 F5 3A 67 EA E6 FE D3 40 F6 01 D3 40
00E0 32 67 EA CD CB 00 28 11 3E 91 CD 29 02 AF 32 C5
00F0 ED 3C 32 CB ED CD FF 00 A7 C1 F5 20 C5 18 C6 3E
0100 00 CD 7C 01 3E 04 32 CB ED 3E 07 CD 7C 01 CD E9
0110 01 47 3E 10 0E 00 57 A0 28 01 0C 7A 07 30 F7 79
  .
  .
8000 E6 F1 E9 F1 A5 7F A5 7F 7F 23 A9 23 A5 7F A5 7F
8010 A5 7F A5 7F A5 7F A5 7F A5 7F A5 7F A5 7F A5 7F
8020 00 2F 80 E8 03 92 20 1C 2C 01 2C 0C FF 8F 00 39
8030 80 F2 03 4F 50 54 F1 13 00 5A 80 FC 03 A0 20 0F
8040 50 2C 0F 19 20 3A 20 9F 20 11 2C 0F 19 20 3A 20
8050 91 20 FF 96 28 0F 0C 29 3B 00 84 80 06 04 91 20
8060 D9 0F 12 F5 4F 50 54 F4 1A 29 20 3B 20 22 4D 41
8070 43 48 49 4E 45 20 43 4F 44 45 20 4D 4F 4E 49 54
8080 4F 52 22 00 A3 80 10 04 97 20 DC 41 24 28 58 29
8090 F1 FF 82 28 22 30 30 30 22 F3 FF 9A 28 58 29 2C
80A0 15 29 00 C4 80 1A 04 97 20 DC 44 24 28 58 29 F1
80B0 FF 82 28 22 30 22 F3 FF 9A 28 FF 97 28 58 29 29
80C0 2C 13 29 00 CC 80 24 04 3A 8F E4 00 D2 80 2E 04
  .
  .
  .
FF50 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
FF60 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
FF70 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
FF80 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FF90 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFA0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFB0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFC0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFD0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFE0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
FFF0 FF 00 FF 00 FF 00 FF FD 0C 00 00 00 00 42 00 00
```

-D0 FFFF ↵ ……ダンプコマンドの実行. 0_H番地から$FFFF_H$番地まで(つまりすべてのメモリ)をダンプする.

0_H番地のメモリの内容

アドレス表示部.

$11F_H$番地のメモリの内容

8000_H番地のメモリの内容

$80CF_H$番地のメモリの内容

$FF50_H$番地のメモリの内容

最終アドレス$FFFF_H$のメモリの内容

この例では1行に16バイトがダンプされている. それぞれの1バイトには, 0_HからFF_H(10進数で255)までの内のひとつの値が記憶されている.

Figure-2.2.1 全メモリ・アドレス0～FFFFHのダンプ

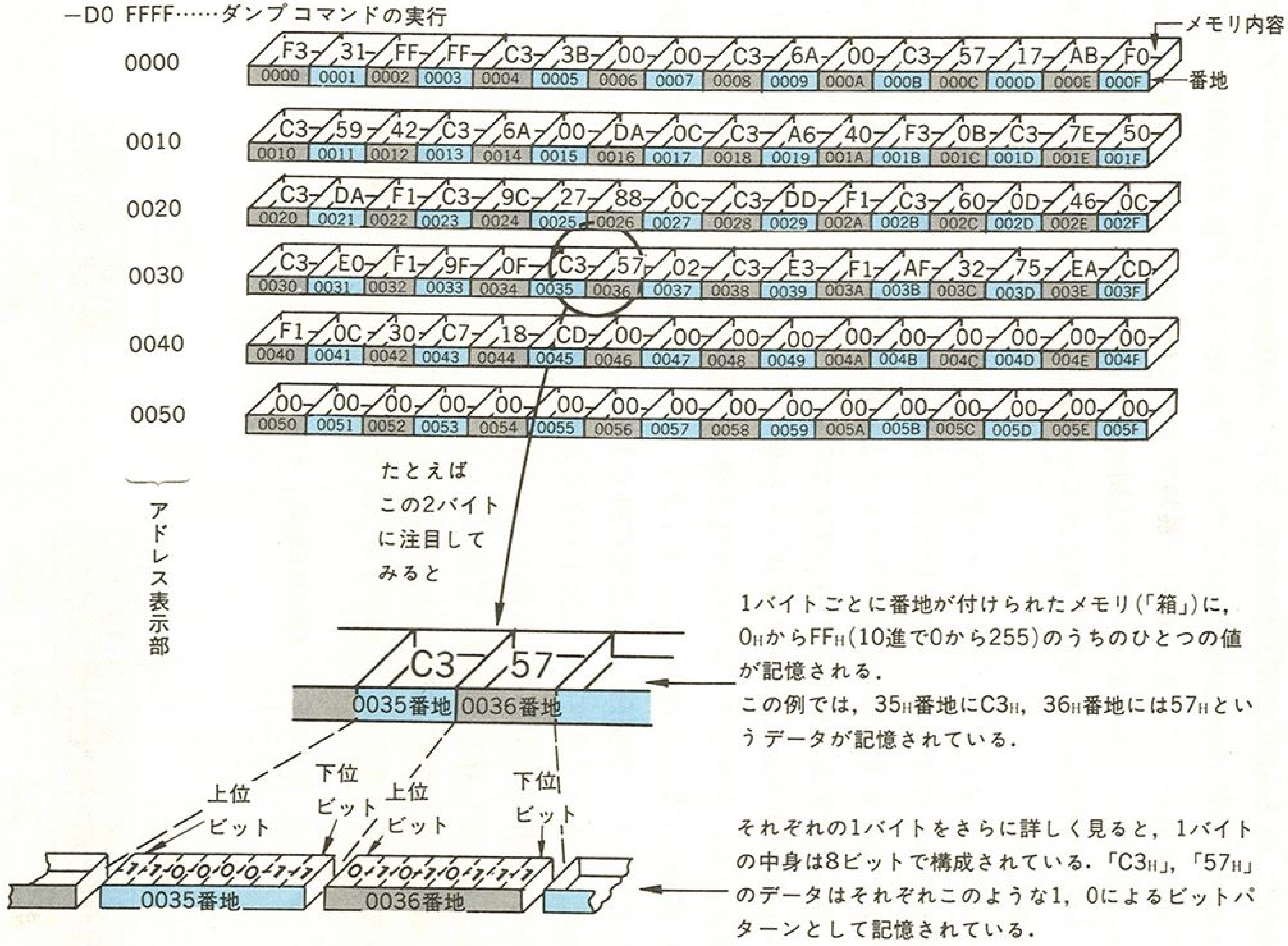


Figure-2.2.2 メモリの概念図の図解

Figure-2.2.1のダンプリストは,PC-8801のN-BASICの例なので，アドレス0_HからはN-BASICのROMの内容，アドレス8000_Hからはユーザー・プログラムエリアのRAMの内容(実はこのエリアにMCモニタ自身のBASICのプログラムがはいっている．後述)，最終アドレス付近はN-BASICが動作するために必要なワーキングエリアのRAMの内容がダンプされています．ただし，メモリのどのエリアが何に使われているかは，ここではあまり問題ではありませんが．

このコマンドを実行して，次々とアドレスやデータが表示されていく様子を眺めているだけでも，メモリについての概念をある程度実感できるのではないかと思います．特に左端のアドレス表示部に注目して，その16進による数の進み具合をよく観察しておいてください．

Figure-2.2.2では，このダンプリストを使って，メモリの概念を再び図解してみました。今度はアドレス0_H付近を使って解説しています．

0_H番地から延々と行なったメモリダンプや，そのリストを利用しての図解などで，メモリの様子がある程度理解できてきたでしょうか．

次に，ひとつのアドレスには1バイトのデータが記憶され，その1バイトのメモリを指し示すアドレス値は2バイトの数値によって表わされていることを，もう少し詳しく解説しておきましょう．

Figure-2.2.1のダンプリストから，たとえばアドレス$806E_H$のメモリ内容は$4D_H$となっていますが，このアドレスとそのデータを例に図解しましょう．

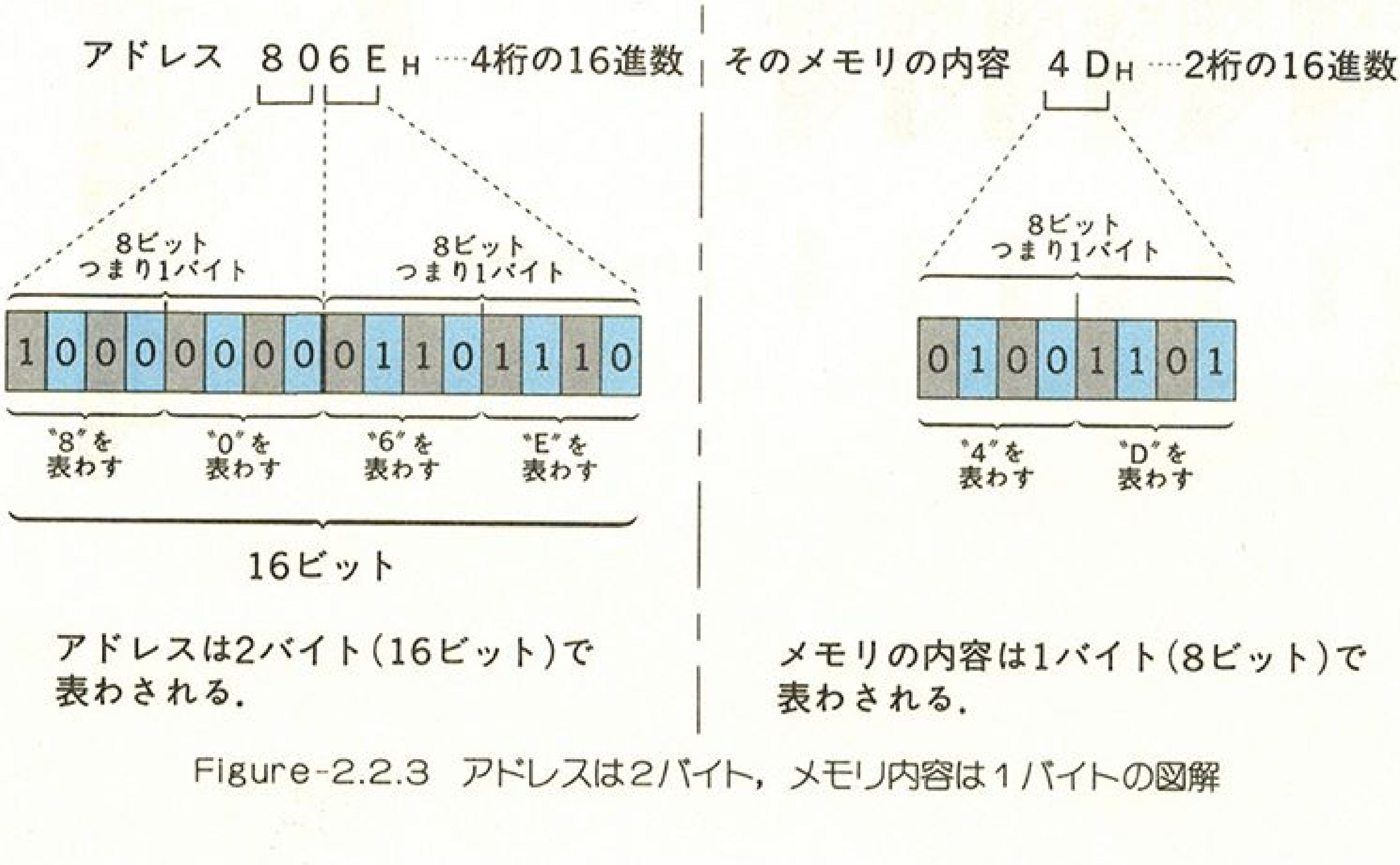


Figure-2.2.3 アドレスは2バイト，メモリ内容は1バイトの図解

前図から，アドレスは2バイトの16ビットにより表わされることがわかると思います．コンピュータは，この16ビットのアドレス情報により，1/1000秒ぐらいの速さで，65536個(65536バイト)の中から該当する一つの番地を捜し出し，リード／ライト（読出し／書込み)の動作を行なうのです．

Z-80や8085，8080，あるいはそれと系統の異なる6502，6800，6809などの一般的な8ビットCPUは，

アドレス：16ビット
データ　：8ビット

で動作しています．つまり，

2バイト(16ビット)のアドレス＝0_H〜$FFFF_H$の16進数(10進数では0〜65535)

1バイト(8ビット)のデータ＝0_H〜FF_Hの16進数(10進数では0〜255)

であり，コンピュータでは2バイト(16ビット)のアドレスにより選択されるメモリ上の1バイト(8ビット)のデータをリード／ライトすることにより，プログラムが実行されているわけです．

このような8ビットのデータを一つの単位として処理するようなマシンを「8ビットのコンピュータ」と言い，その2倍の16ビットまでのデータを一つの単位として処理できるマシンを「16ビットのコンピュータ」と言います．

このあたりの動作のしくみは，4章以後でさらに詳しく解説していきます．

9.2章で「アスキー表示」とか「キャラクタコード」などについて，応用プログラムを作って解説しますが，MCモニタのダンプコマンドは，実は同時にアスキー表示（Figure-2.2.5参照)もできるのです．MCモニタのプログラムの1310行の「 ′」を削除すれば，次の例のように，左側の16進数値をキャラクタコードとみた場合の文字が，右側に表示されます．巻末のキャラクタコード表を見ると，20_H未満の値のコードには，「文字」はありませんのでこの場合にはピリオド「.」を表示します．

```
1270 FOR I=MSTART TO MEN STEP OPT*8
1280   PRINT FNA$(I);" ";
1290   FOR Q=0 TO OPT*8-1
1300     PRINT FND$(I+Q);" ";
1310     'GOSUB 1620
1320     IF I+Q=MEN THEN Q=OPT*8
1330   NEXT
1340   PRINT
1350 NEXT
```

1310行の「'」：この部分を削除する

Figure-2.2.4 Dコマンドにアスキー表示部を付ける

C H I N E ␣ C O D E

```
8000 E6 F1 E9 F1 A5 7F A5 7F 7F 23   ◥円♥円・ ・  #ｩ#・ ・
8010 A5 7F A5 7F A5 7F A5 7F A5 7F   ・ ・ ・ ・ ・ ・ ・ ・
8020 00 2F 80 E8 03 92 20 1C 2C 01    /_♠ ┤   ,  ,   + 9
8030 80 F2 03 4F 50 54 F1 13 00 5A   _年 OPT円  Z_
8040 50 2C 0F 19 20 3A 20 9F 20 11   P,    : ﾉ   ,   :
8050 91 20 FF 96 28 0F 0C 29 3B 00   ┬  |(   ); ▄_  ┬
8060 D9 0F 12 F5 4F 50 54 F4 1A 29   ﾙ  ♠OPT日 ) ; "MA
8070 43 48 49 4E 45 20 43 4F 44 45   CHINE CODE MONIT
8080 4F 52 22 00 A3 80 10 04 97 20   OR"  ｣_   | ﾜA$(X)
8090 F1 FF 82 28 22 30 30 30 22 F3   円 _("000"月 ﾚ(X),
80A0 15 29 00 C4 80 1A 04 97 20 DC    ) ﾄ_   | ﾜD$(X)円
80B0 FF 82 28 22 30 22 F3 FF 9A 28    _("0"月 ﾚ(  K(X))
80C0 2C 13 29 00 CC 80 24 04 3A 8F   , ) ﾌ_$ :+◢ ﾒ_.
80D0 91 00 E6 80 38 04 AF 20 85 20   ┬ ◥_8 ｯ ■ "-";BU
80E0 46 46 45 52 24 00 00 81 42 04   FFER$  _B COMMND
80F0 24 F1 FF 81 28 42 55 46 46 45   $円 _(BUFFER$, )
```

左の16進数値に対する文字が表示される．

アスキー表示部

PC-8001，PC-8001 MKII，PC-8801 のN-BASICでは，アドレス 8000H 以後はユーザー・プログラムの格納エリアにあたる．この場合はMCモニタのプログラムが格納されている状態がダンプされている．*

Figure-2.2.5 アスキー表示部が付いたDコマンド表示例

*PC-6001用のMCモニタ（巻末APPENDIX 1）には，アスキー表示の機能はない．

3

2進法と16進法

コンピュータの内部は，すべてのデータが「0」か「1」かのビットの組合せで表現される2進の世界です．コンピュータの中のどこを覗いてみても「数」は0と1しか存在しません．CPUの中もメモリの中も0と1しかないのです．(実際は，電圧が「ある」か「ない」かで0，1の区別をしている)このことは，2章でもいくつかの図解で説明しました．しかし，0と1の組合せでは，コンピュータにはわかっても私たち人間にはピンときません．たとえば「001111101000」と書いてもちょっと見当がつきませんね．実はこの1，0の組合せは，10進の「1000」を表わしています．

私たち人間には，10進を使うのが一番馴じみ深くて理解しやすいのですが，残念ながら10進はコンピュータの内部を表わすには適していません．そこで16進の登場となるわけです．16進がなぜ適しているのでしょう．それは2進の8ビット，つまり1バイトのデータを2桁でうまく表現できるからなのです．

「2進」，「8ビット」，「16進」の3者は，同時に理解しなければ意味がないほど密接な関係にあります．本章ではこの3者を関連づけて，コンピュータに即した解説を行なっていきます．

3.1

●16進の予備知識

現在の多くのコンピュータは，マシン語やアセンブラレベルでものを考えるとき，数値を16進で数えます．ひと昔前は8進を使ったものもありましたが（8ビットのマイクロコンピュータが世に出てしばらくは，8進を使うアセンブラが主流であった），今は特殊なものでしか使われていません．

まずは16進での数の数え方から解説していきましょう．16進で使う「数字」は次の16種類です．（　）の中は10進での値です．

0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	A,	B,	C,	D,	E,	F
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)

この0からFまでの「数字」を使って16進で数を数えるわけですが，とりあえず0から順に数えていってみましょう．（　）の中は同じ数を10進で表わしたものです．

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
(16)	(17)	(18)	(19)	(20)	(21)	(22)	(23)	(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
(32)	(33)	(34)	(35)	(36)	(37)	(38)	(39)	(40)	(41)	(42)	(43)	(44)	(45)	(46)	(47)
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
(48)	(49)	(50)	(51)	(52)	(53)	(54)	(55)	(56)	(57)	(58)	(59)	(60)	(61)	(62)	(63)
⋮															⋮

⋮															⋮
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
(144)	(145)	(146)	(147)	(148)	(149)	(150)	(151)	(152)	(153)	(154)	(155)	(156)	(157)	(158)	(159)
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
(160)	(161)	(162)	(163)	(164)	(165)	(166)	(167)	(168)	(169)	(170)	(171)	(172)	(173)	(174)	(175)
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
(176)	(177)	(178)	(179)	(180)	(181)	(182)	(183)	(184)	(185)	(186)	(187)	(188)	(189)	(190)	(191)
⋮															⋮
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
(240)	(241)	(242)	(243)	(244)	(245)	(246)	(247)	(248)	(249)	(250)	(251)	(252)	(253)	(254)	(255)
100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
(256)	(257)	(258)	(259)	(260)	(261)	(262)	(263)	(264)	(265)	(266)	(267)	(268)	(269)	(270)	(271)
⋮															⋮
FFF0	FFF1	FFF2	FFF3	FFF4	FFF5	FFF6	FFF7	FFF8	FFF9	FFFA	FFFB	FFFC	FFFD	FFFE	FFFF
(65520)	(65521)	(65522)	(65523)	(65524)	(65525)	(65526)	(65527)	(65528)	(65529)	(65530)	(65531)	(65532)	(65533)	(65534)	(65535)

途中で省略してありますが，ここは同様に数えていくこととして$FFFF_H$まで数えてみました．MCモニタのダンプ・コマンドによるアドレス部の表示とまったく同じ数え方ですね．

0からFまでの数字を使う16進の数え方は，だいたいわかったのではないかと思います．この16進を使うと，10進の0から15(16進でF_H)までの数が1桁で表わせ，同様に255(16進でFF_H)までが2桁で表わせることに注目しておいてください．

8ビット(つまり1バイト)の0と1の組合せで表現できる範囲の数値は，16進を使うとちょうど2桁ですべてを表わせることになります．つまり16進は，2進と8ビットを取り扱う上でたいへん都合のよい数え方であるわけです．

16進の数え方に少し慣れたところで，次に2進と8ビットについて解説し，それらと16進との関係を明らかにしていきましょう．

3.2

●2進,8ビット,16進の関連

2進と8ビット，それに16進の3者は密接に関連しており，別々に考えることはできません．まずは，2進と8ビットについての解説から始めましょう．

8ビット，つまり1バイトを普通は次のように図示します．

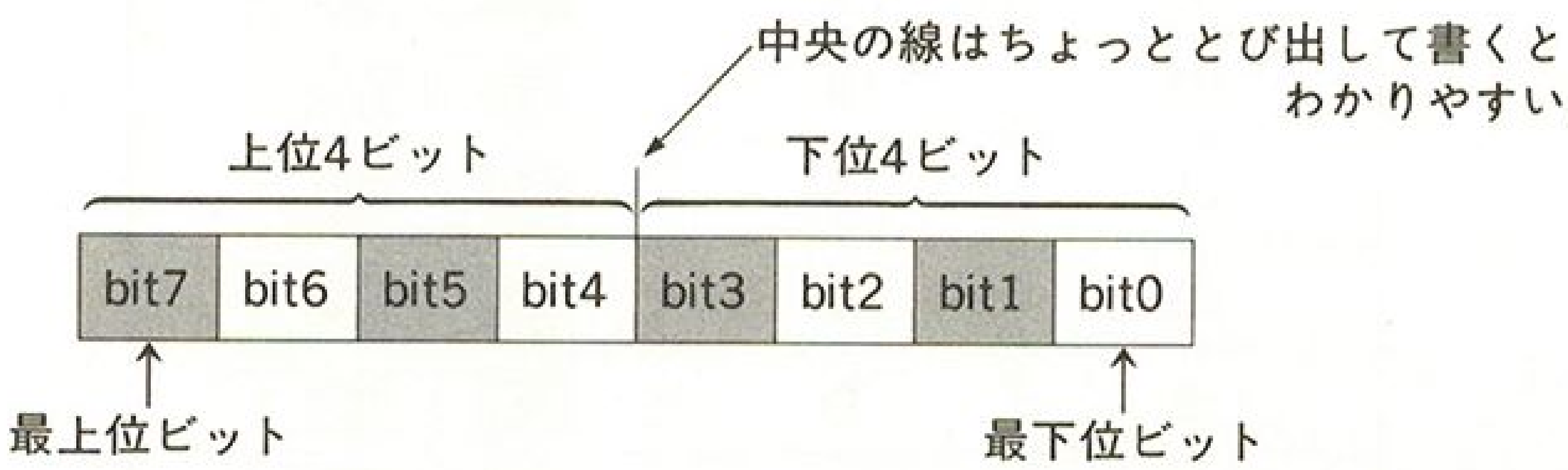


Figure-3.2.1　1バイトのビットパターン図示

この図のようにそれぞれのビットは，最下位ビットから，bit 0，bit 1，bit 2，‥‥bit 7という具合に呼ばれます．また，16進で考える場合は，8ビットを真中から4ビットずつに分けて，それぞれを16進の1桁として扱います．つまり8ビットは16進の2桁で表わされ，16進の1桁は2進の4ビットで表わされるのです．

次に，16進の1桁（0からFまで）を4ビットで表わしたものを示しましょう．10進数とも対応させて，10進，16進，2進の対照表の形にしておきました．

10進	16進	2進の4ビット			
		bit$^{3}_{7}$	bit$^{2}_{6}$	bit$^{1}_{5}$	bit$^{0}_{4}$
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	A	1	0	1	0
11	B	1	0	1	1
12	C	1	1	0	0
13	D	1	1	0	1
14	E	1	1	1	0
15	F	1	1	1	1

Figure-3.2.2　10進，16進，2進の対照表

さて，Figure-3.2.1に示した，1バイトのビットパターンの図に従って，Figure-2.2.1のダンプリストから適当なデータを図示してみましょう．Figure-2.2.2(メモリの概念の図解)でも取り上げたアドレス0035_Hのメモリ内容は$C3_H$ですが，このビットパターンは次のようになります．対照表とも照らし合わせてください．



Figure-3.2.3 16進の「C3」のビットパターン

この16進と2進の関係は，次のように考えると簡単に理解できます．

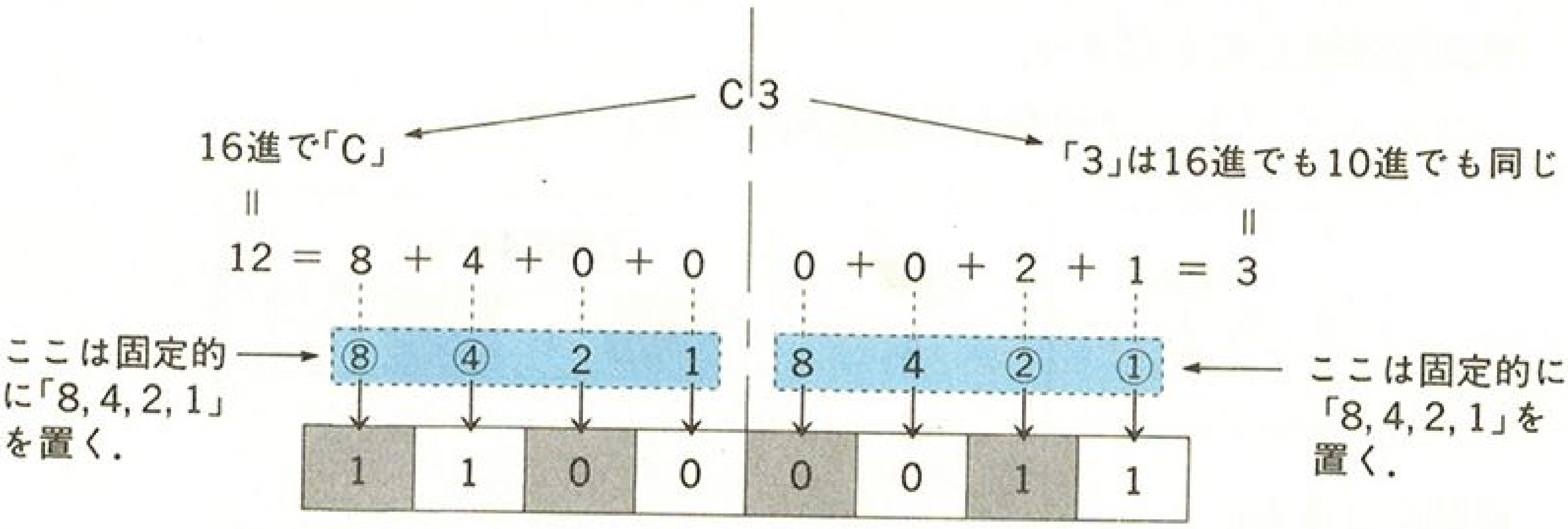


Figure-3.2.4 16進 → 2進への変換

この図をじっと見ているだけでも何かがわかってくるかも知れません．

ここのところは非常に大切なので詳しく解説しておきましょう．

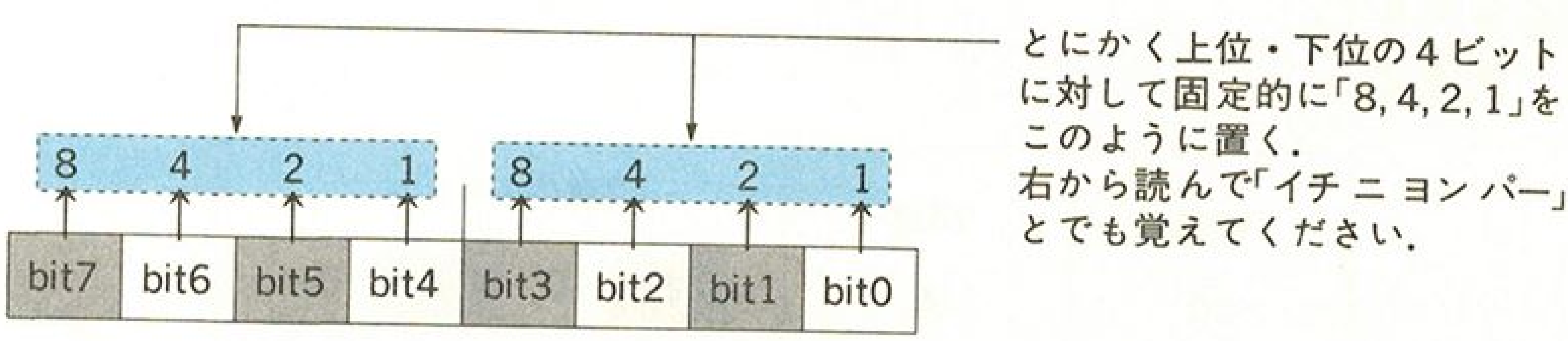


Figure-3.2.5 2進と16進との関係

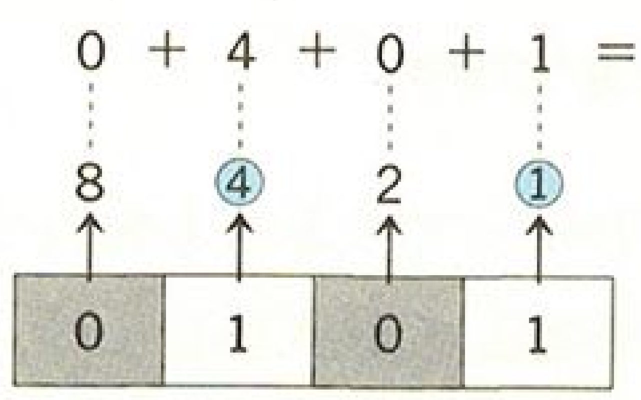
たとえばビットパターンがわかっている場合，1バイトの8ビットを上位4ビット，下位4ビットに分け，それぞれのビットに前図のように8，4，2，1の数値を固定的に持たせます．そしてビットパターンの「1」が立っているビットのところの数値だけを4ビットごとに合計します．その合計値が4ビットのビットパターンで表わされたものの10進での値となります．また，10進を2進のビットパターンに落すには，この逆の操作を行なえばよいわけです．

これらの具体例を2，3示しましょう．Figure-3.2.2の10進，16進，2進の対照表を参照してください．

たとえば，「5」は10進も16進も同じ「5」ですが，

10進	16進	2進の4ビット			
5	5	0	1	0	1

を例にとると，

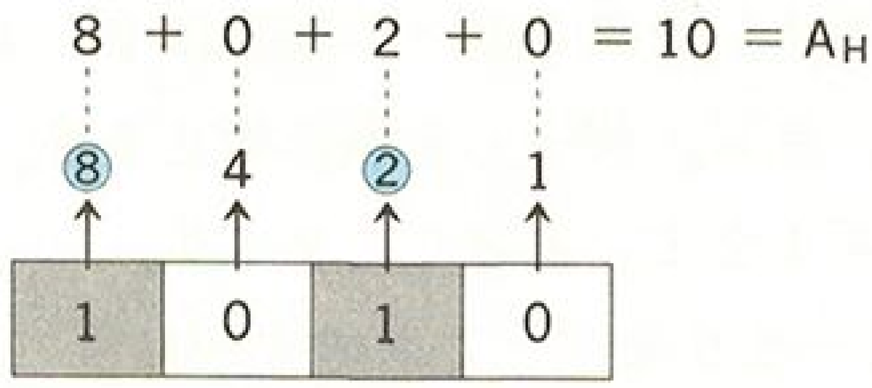


となります．

同様に

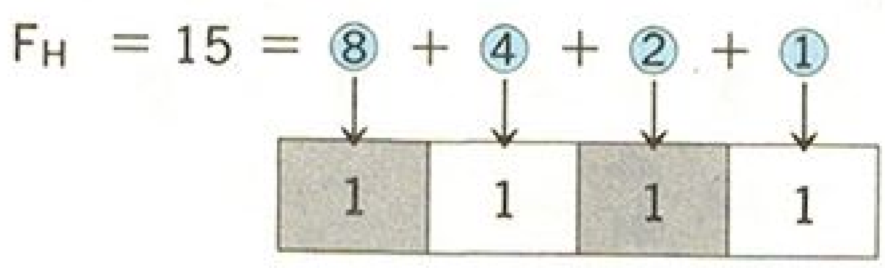
10進	16進	2進の4ビット			
10	A	1	0	1	0

を例にとると，



となるわけです．

また，16進の「F」のビットパターンはどうなるでしょう．



このように，すべてのビットに1が立つことになります．

もう一度この10進，16進，2進ビットパターンの変換方法の原理をまとめておきましょう．16進→2進，2進→16進への変換方法を区別する必要はありません．本章で解説している16進と2進との関係さえ理解できれば，両者は同じ問題であることがわかります．

まず，すべての数値を16進の1桁，つまり2進の4ビットに分けて考えます．

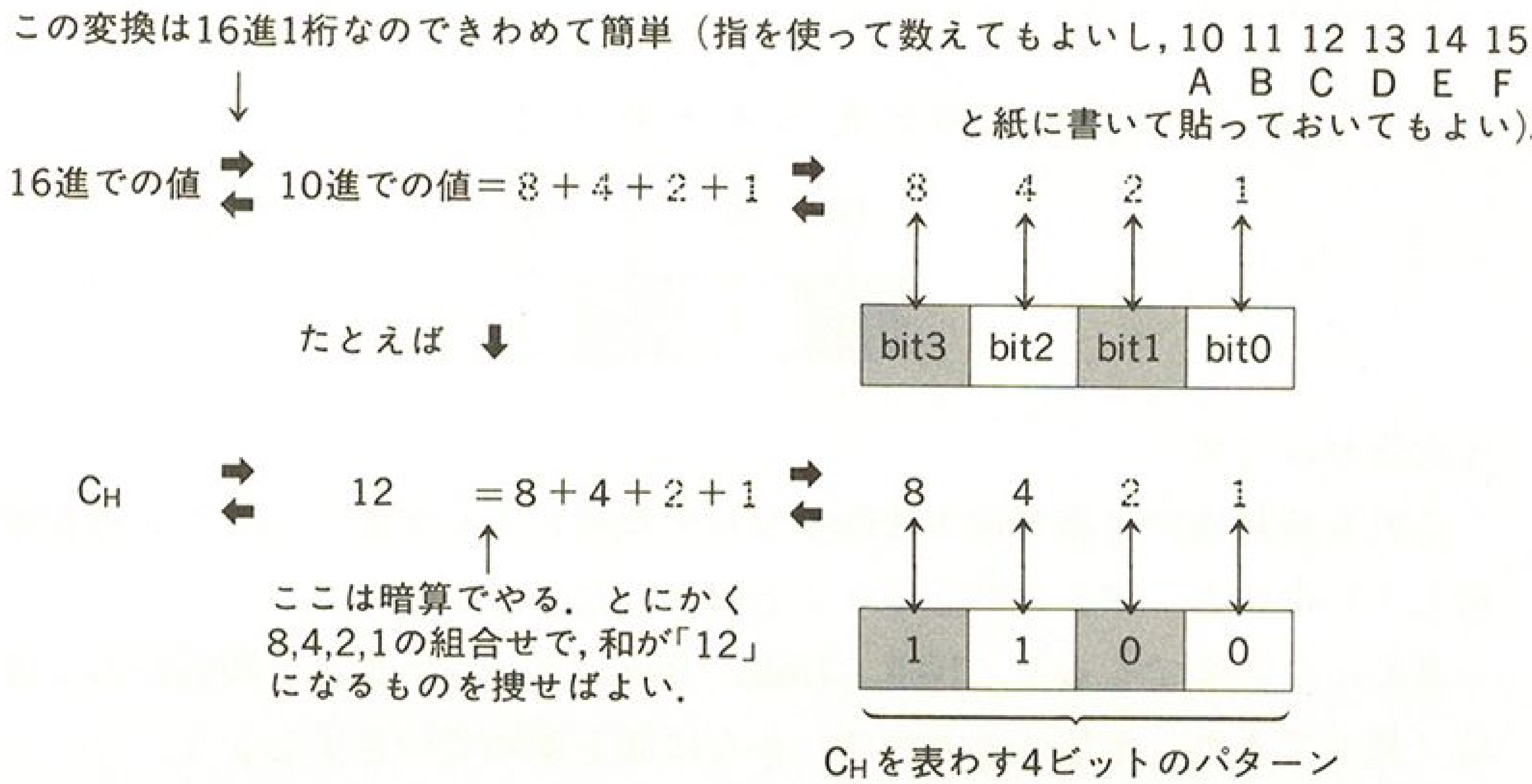
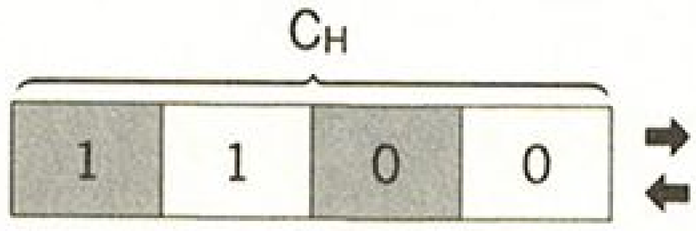


Figure-3.2.6　2進，16進，ビット・パターン間の変換方法

このようにすべては16進の１桁で考え，その２進への変換は８，４，２，１つまり右から読めば「イチ，ニー，ヨン，パー」を基に考えます．これがすべてです．まず第一に紙の上に「８４２１」と書くことです．

多くの参考書が解説している形式(つまりは同じことですが)，



$$\rightleftarrows 2^3\times1+2^2\times1+2^1\times0+2^0\times0$$
$$=8+4+0+0$$
$$=12 \rightleftarrows C_H$$

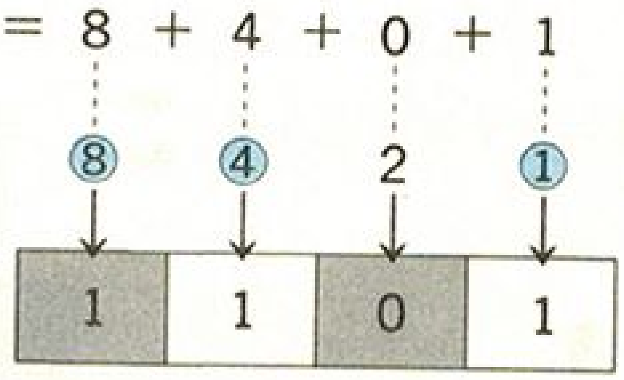
のようなめんどうなことをする必要はありません．すべては８，４，２，１で片付けられます．

もうこの10進↔２進の変換方法はわかりましたね．10進や16進を２進に落すことも簡単にできるようになったと思います．

その方法は16進ならばまず10進になおしたあと，８，４，２，１のどれを取り上げて合計すれば10進の値と一致するかを考えればいいのです．取り上げたビットの位置に１を置き，残りを０とします．

たとえば16進の「D」であれば，

$$D_H=13=8+4+0+1$$

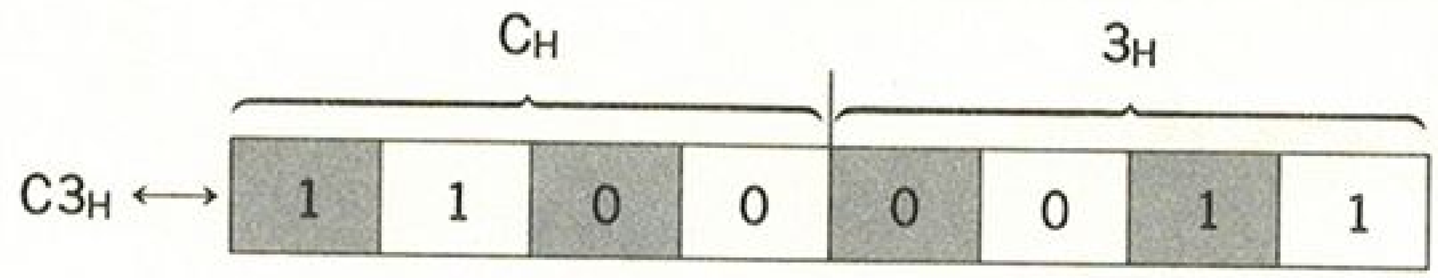


となるわけです．

これらの10進↔２進変換方法はかならず理解してください．ここら辺を理解していないと，マシン語を扱うことはまったくできません．

さて，ここまでくると，10進，16進，２進，８ビットなどの関係がある程度「見えてきた」と思いますので，さらに話を進めていきましょう．

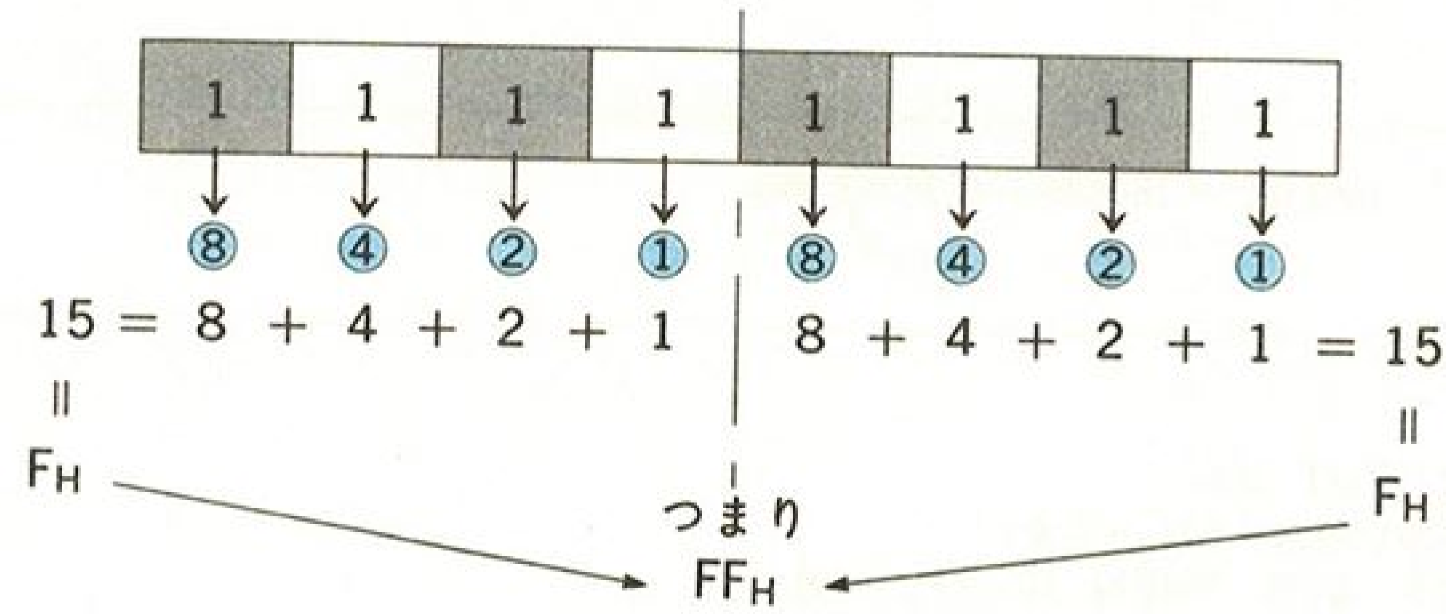
ここでもう一度Figure-3.2.4に帰ってみてください．



となることはもう図を見るだけで理解できることでしょう．

もう一例だけ示しておきましょう．1バイトで表わせる最大値であるFF_H(10進で255)の場合です．

FF_Hは8ビットで表わせる最大値ですから，すべてのビットに1が立っているわけですから，次の図のようになります．



というわけです．

本章から，2進，10進，16進の数え方や，8ビットでの数の表わし方などが，ある程度明らかになったことと思います．また，マシン語を扱う場合，16進を利用すると都合がよいことも納得できたでしょう．なお複数桁の10進↔16進変換については，みなさんのBASICマシンで簡単に行なえる方法をこのあとに紹介します．

本章はかならず理解しなければならない重要な部分です．しかし完全に理解はできなくても，さきに進んで結構です．もし本章に関連したところでつまずいたときには，また戻ってくればよいでしょう．本章がだいたい理解できれば，マシン語を学ぶのは難しくありません．さきは非常に明るいと思って進んでください．

ここで参考までに，BASIC上で簡単に10進数↔16進数の変換を行なう方法を示しておきましょう．

16進数　10進数

プログラムを組む必要はなく，*
ダイレクト実行でよい．

```
Ok
PRINT HEX$(15)
F
Ok
PRINT HEX$(16)
10
Ok
PRINT HEX$(100)
64
Ok
PRINT HEX$(1024)
400
Ok
PRINT HEX$(65535)
FFFF
Ok
```

Figure-3.2.7　10進数 → 16進数の変換実行例

16進数⇨10進数変換プログラム

```
10 INPUT A$
20 A=VAL("&H"+A$)
30 IF A<0 THEN A=A+2^16
40 PRINT A
50 GOTO 10
```

その実行例

16進数　10進数 →

```
Run
? 0A
 10
? 0F
 15
? 10
 16
? FF
 255
? 3FF
 1023
? C000
 49152
? FFFF
 65535
?
```

Figure-3.2.8　16進数 → 10進数の変換プログラムと実行例

＊HEX$関数のない機種の場合は，自分でプログラムを作る必要がある．巻末の APPENDIX 1 にある PC-6001用MCモニタ・プログラムのリスト2010行〜2080行を参照．

4

コンピュータに直接命令できるのはマシン語だけ

本章では，コンピュータとマシン語の関係を，BASIC言語を起動している状態のコンピュータを例にとって解説します．また，非常に簡単なマシン語のプログラムをMCモニタを使って入力し，実行してみます．そこからCPUというものの存在を認識し，次のステップへの足掛りとしましょう．

おそらくこの時点では，コンピュータとマシン語との関係は適確には理解し難いと思われます．というのはこの「関係」とは単純なものではなく，コンピュータ・アーキテクチャの本質に迫る問題なのです．本書を読み終える頃にもう一度本章を読み直してみてください．ここで言っていることがさらによくわかると思います．

4.1

●BASICはマシン語のプログラム

私たちの多くは，BASIC言語を使ってプログラムを書き，いろいろな仕事をコンピュータにさせています．そしてBASICユーザーの中には，BASICで書いたプログラムが直接コンピュータを働かせているものと思っている人も多いでしょう．電源ONで即BASICが使えるマシンがほとんどなので，そう思うのは無理もないことですが，実はそうではありません．

BASICで書いたプログラムは，結果としてコンピュータを働かせますが，それはあくまで結果であり，BASICで書いたプログラム自身ではけっしてコンピュータは働きません．

たとえば，

```
PRINT "ABC"↵
```

と入力すれば，

```
ABC
```

とスクリーンに表示されますが，この「PRINT "ABC"」というプログラムは，コンピュータそのものにはまったく通じないのです．「コンピュータはマシン語でしか働かない」ということは，しごく当然のことで，コンピュータを扱う人なら誰でも知っているようですが，実はまだまだ理解されていないようです．

ここで次のことをはっきりと頭に入れ，認識を新たにしてください．

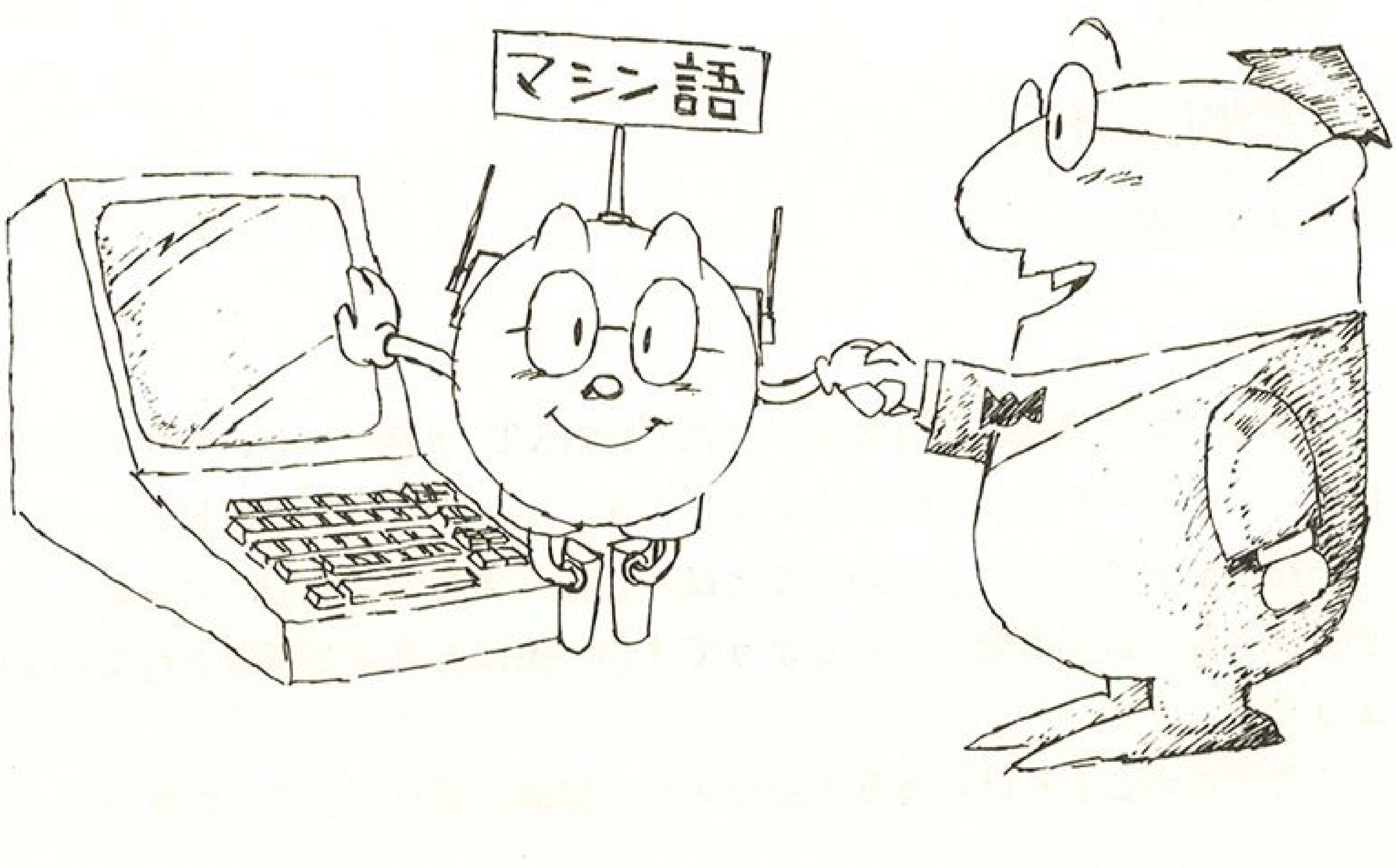
コンピュータ＝BASICマシンではない.
コンピュータはマシン語でしか働かない.

これはコンピュータを理解する上で非常に重要なことであり，コンピュータの本質に迫ることなのですが，本書を読み進んでいくうちにしだいに明らかになっていくでしょう.

コンピュータはマシン語しか理解できません．だから「PRINT "ABC"」というような人間の言葉に似た文章などを入力しても働くわけはないのです.

でも事実としてBASICで書いたプログラムが働いているのはなぜなのか．ここでその説明をしておきましょう.

BASICで書いたプログラムが働くのは，コンピュータがBASICインタープリタ*というマシン語でできているプログラム(BASIC言語を使えるようにするための，マシン語でできているプログラム)を実行中であるからです.



*BASIC言語で書かれたプログラムをマシン語に変換するプログラム．インタープリタとは命令の一つひとつをマシン語に変換しながら実行するもので，実行速度はコンパイラの方が格段に速い.

コンピュータの電源をONにしたとき，多くのパーソナル・コンピュータの場合は，ROMに書き込まれているBASICインタープリタが自動的に実行され，BASIC言語が使えるようになります．この状態が次に示すようなオープニング・メッセージが表示され，OKプロンプトが出力されている状態なのです．



Figure-4.1.1 BASICインタープリタが実行されている状態

ここで重要なことは，この状態で私たちはコンピュータを表面的にはBASICマシンとして扱っているのですが，内部では，BASICインタープリタというマシン語のプログラムが常に実行され続けているということです．

ここでBASICインタープリタの仕組みを少し説明しておきましょう．

PRINT "ABC" ↵

と入力すると，BASICインタープリタは，「PRINT "ABC"」という文字列を分析して解読します．解読できなければ「××ERROR」ということになりますが，成功すれば直ちに" "内の文字をスクリーンへ表示するためのマシン語のプログラム(いろいろな機能のマシン語のプログラムを，ライブラリーとしてBASICインタープリタの中に持っている)をライブラリーの中から捜し出して実行するわけです．

一般のパーソナル・コンピュータで使われているBASICインタープリタとは，次のように言えばよいでしょう．

BASIC言語の文法に従って書かれたユーザー・プログラムを解読して，自分自身で用意しているいくつかのマシン語プログラムに置き換える作業を1行ごとに行ないながら，置き換えたマシン語を実行していく機能を持った，マシン語でできているプログラムである．

何だかややこしくなりましたがわかりますか？　ここのところを図示してみましょう．

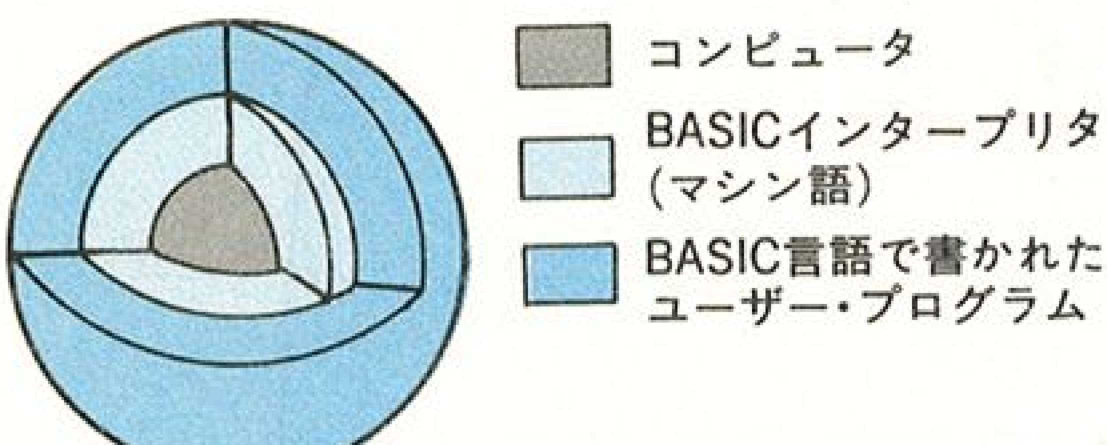


Figure-4.1.2　一般のパーソナル・コンピュータでBASICを使用している状態

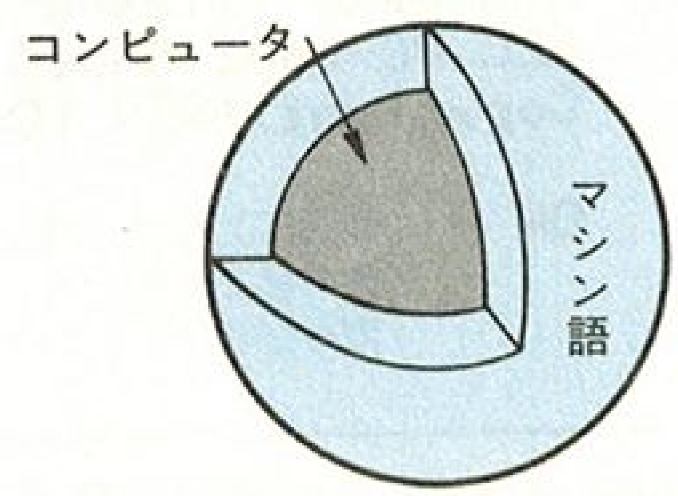


Figure-4.1.3　コンピュータはマシン語で包まれています．

この図は中身が覗けますので，核となっているコンピュータの層が見えていますが，通常はこの表面しか見えないわけです．つまり私たちがパーソナル・コンピュータに組込みのBASICを使っているのは，この表面の「BASIC言語で書かれたユーザー・プログラム」だけをさわっているにすぎないのです．

電源を入れれば自動的にBASICが起動して，この図のようにコンピュータはBASICの層に包まれてしまいます．多くのBASICユーザーが，「パソコン＝BASICマシン」という認識しか持っていないというのは，もっともなことでしょう．

コンピュータに直接触れることができるのはマシン語のみです．とにかくコンピュータはマシン語でなければ動かすことはできません．

BASICインタープリタとマシン語との話が長くなってしまいましたが，この辺でマシン語そのものの解説にはいっていきましょう．

4.2

●マシン語でコンピュータを直接働かせてみる

コンピュータの構成やら，CPUの働きなどの解説を行なう前に，とにかく一度,簡単なマシン語のプログラムをコンピュータに実行させてみましょう．

コンピュータは，メモリ上のマシン語のプログラムによって，そのプログラムが指示する通りの仕事を忠実に実行します．メモリ上にロードされたマシン語のプログラムは，BASIC言語のような文章形式のプログラムとはまったく異なり，00_H～FF_Hの間の数値の羅列です．

これらの数値がコンピュータに対する各種の命令を表わしたり，アドレスを表わしたり，データとしての数値を表わしたりするわけです．

何はともあれ，マシン語のプログラムをMCモニタを使ってメモリ上に書き込んで実行してみましょう．

ここに12バイトから成るマシン語の小さなプログラムを用意しました．まずこれをMCモニタを起動し，そのSコマンドを使って，メモリに書き込んでください(機種によってはマシン語を書き込むためのメモリ上のフリーエリアが異なるものもあります．巻末のAPPENDIX 1「各機種でMCモニタを利用する場合の注意点」を参照してください)．

このプログラムは次の図に示すように,アドレス0000_Hから100バイト分,つまりアドレス0000_H～0063_Hの間のメモリ内容を，アドレス$A000_H$(機種によっては$E000_H$)からの100バイトにそっくりコピーする(このことを「ブロック転送」と言う)ものです．

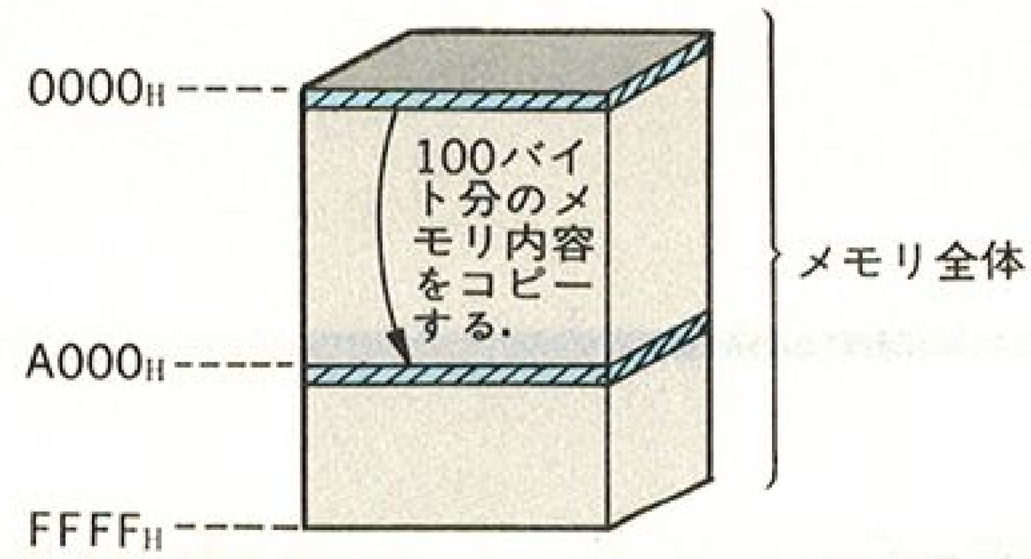


Figure-4.2.1 「12バイトの小さなマシン語プログラム」の働き.

この12バイトをSコマンドによりメモリに書き込む例を次に示します(機種によっては9をDに，AをEに変更).

```
              ここは書き換え前のメモリ内容なので，コンピュータの使用状態によって異なる.*
-S9000↵ ………Sコマンドの実行. アドレス 9000Hからメモリへの書込みを行なう.
9000 00 21↵ ……アドレス9000 Hに21 Hを書き込む.
9001 FF 00↵ ……アドレス9001 Hに00 Hを書き込む.
9002 00 00↵ ……アドレス9002 Hに00 Hを書き込む.
9003 FF 11↵ ……アドレス9003 Hに11 Hを書き込む.
9004 00 00↵
9005 FF A0↵
9006 00 01↵
9007 FF 64↵
9008 00 00↵
9009 FF ED↵
900A 00 B0↵
900B FF C9↵……… 計12バイトのデータを書き込む.
900C 00 . ↵……… ピリオドでSコマンドを終了する.

-D9000 900B↵……Dコマンドで書き込まれたデータの確認.
9000   21 00 00 11 00 A0 01 64 00 ED B0 C9
       アドレス9000 Hから12バイトの小さなプログラムが書き込まれている.
-
```

Figure-4.2.2 マシン語の小さなプログラムをMCモニタを使ってメモリに書き込む

アドレス9000H(機種によってはD000H)から12バイトのデータが正しく書き込まれたか確認したあと，このプログラムを実行します．マシン語のプロ

＊9000H番台のエリアをフリーエリアとして使用できない機種では，マークされている9をDに，AをEに変更する必要がある．以後のリストも同様．

グラムを実行するときは,何番地から実行するかを示す実行開始アドレスをかならず指定しなければなりません.この例では9000$_H$(機種によってはD000$_H$)が実行開始アドレスですのでMCモニタのGコマンドを次のように実行します.

```
-G9000↵ ………Gコマンドで,アドレス9000Hからのマシン語プログラムを実行する.
-  ………実行が終わってMCモニタに戻っている.
```

Figure-4.2.3 Gコマンドでマシン語プログラムを実行する

マシン語プログラムの実行が終わると(瞬時に終わる)再びMCモニタのプロンプト「-」に戻ります.

これでさきほどメモリに書き込んだ12バイトの短いマシン語のプログラムが実行されました.アドレス0000$_H$から100バイト分のメモリ内容が,アドレスA000$_H$からの100バイトにブロック転送されているはずです.その結果をDコマンドでダンプして確認してみましょう.

実行例を次に示します.

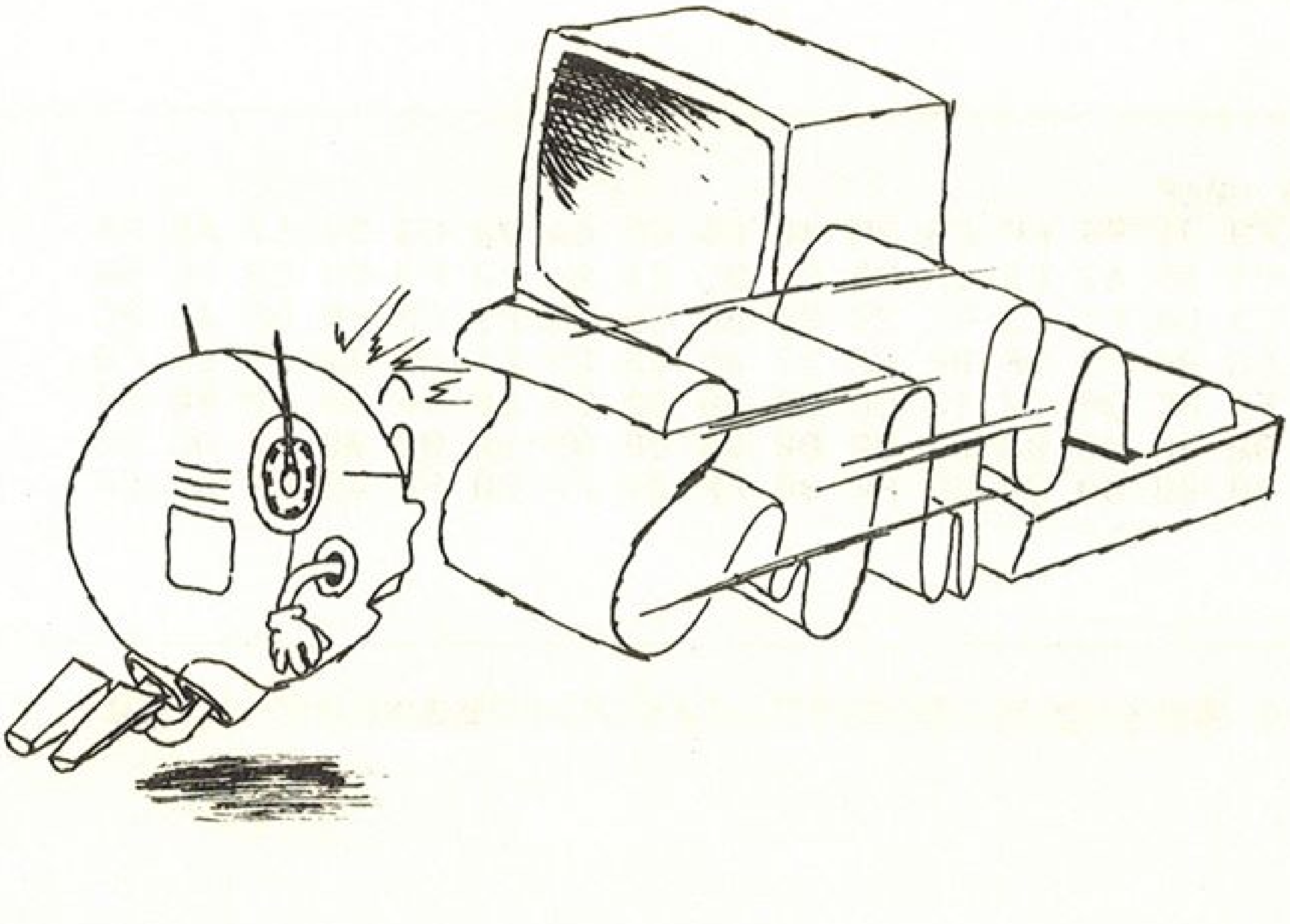
```
-DA000 A06F↵ ………DコマンドでアドレスA000H ～A06FH をダンプする.
A000  F3 31 FF FF C3 3B 00 00 C3 6A 00 C3 57 17 AB F0
A010  C3 59 42 C3 6A 00 DA 0C C3 A6 40 F3 0B C3 7E 50
A020  C3 DA F1 C3 9C 27 88 0C C3 DD F1 C3 60 0D 46 0C
A030  C3 E0 F1 9F 0F C3 57 02 C3 E3 F1 AF 32 75 EA CD
A040  F1 0C 30 C7 18 CD 00 00 00 00 00 00 00 00 00 00
A050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A060  00 00 00 00|00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
      ここまでが ←┘
-     100バイト
```

Figure-4.2.4 実行されたマシン語プログラムによりブロック転送されたデータの確認

このダンプリストと，Figure-2.2.1のダンプリストのアドレス0000_H～0063_Hの内容とを比較してください．12バイトの短いマシン語のプログラムの実行により，この100バイトのデータがそっくりアドレス$A000_H$からのメモリにコピーされていることが確認されます．

本書の実行例はNECのPC-8801のN-BASICモード上で行なっていますので，アドレス0000_H～$7FFF_H$はROMエリアです．よって今回のブロック転送は，ROMの内容の一部をRAM上にコピーしたということになります．

まだこの段階では，コンピュータの内部構成や，CPUの働き，マシン語とアセンブラの関係などは解説していませんので，ここでは単にメモリに短いマシン語のプログラムを書き込んでそれを実行した，ということだけで結構です．



5

CPU，メモリ，I／O，バス

マシン語を学ぶことはコンピュータのアーキテクチャを理解することである，と1章で述べましたが，いよいよ本章から具体的にこのアーキテクチャの核心部の解説にはいっていきます．

マシン語は，CPU，メモリ，I/Oなどに密着した働きをしますので，これらのハードウェアの基本的な知識は必要不可欠なものと言えます．本章では，まずコンピュータ・システムの構成について解説し，順にCPU，メモリ，I/O，バスの解説を行ないます．

5.1

●コンピュータのシステム構成

まず，8ビットのパーソナル・コンピュータなどのコンピュータ・システムの構成について解説しましょう．コンピュータの基本的なシステム構成は非常にシンプルです．

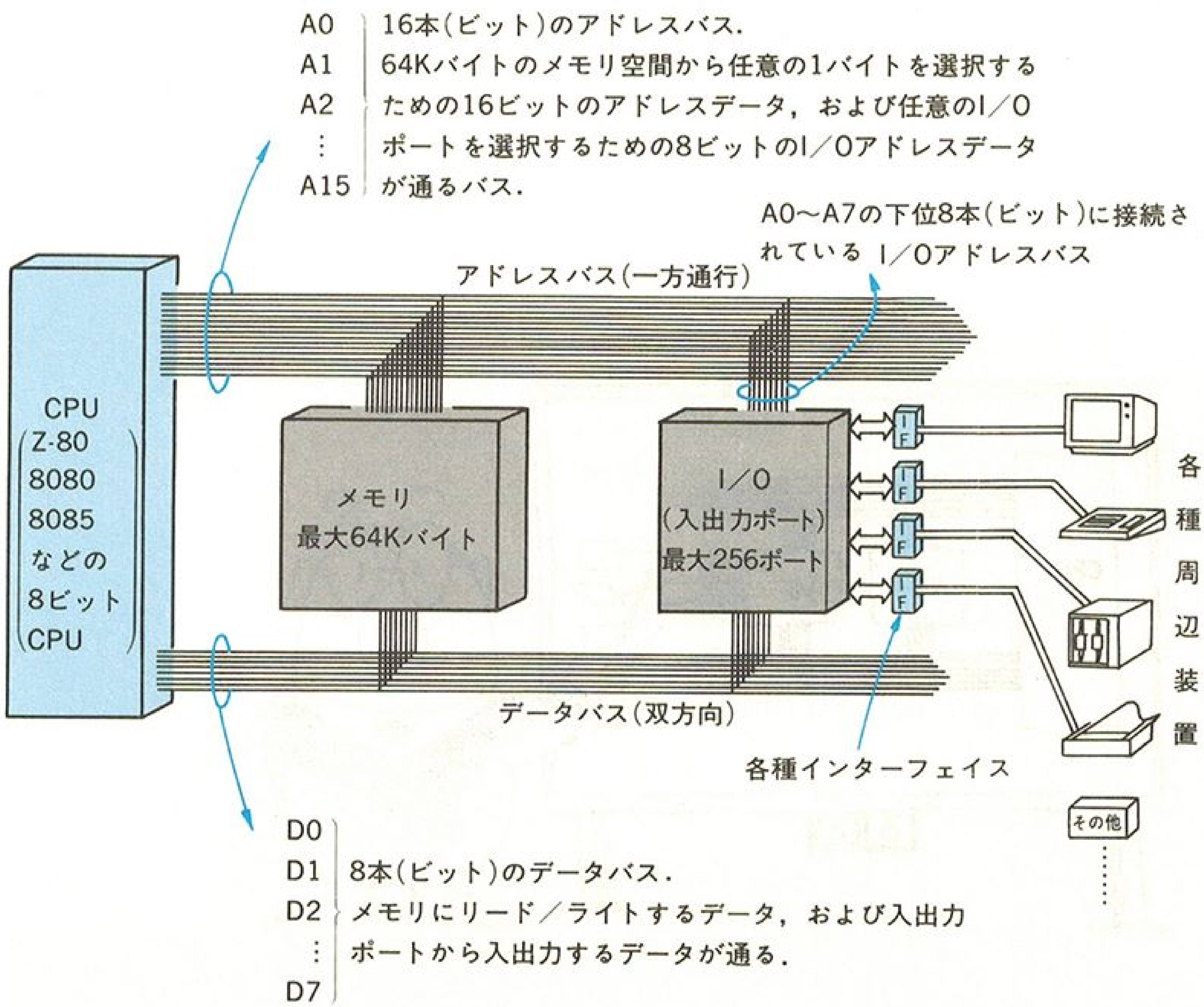
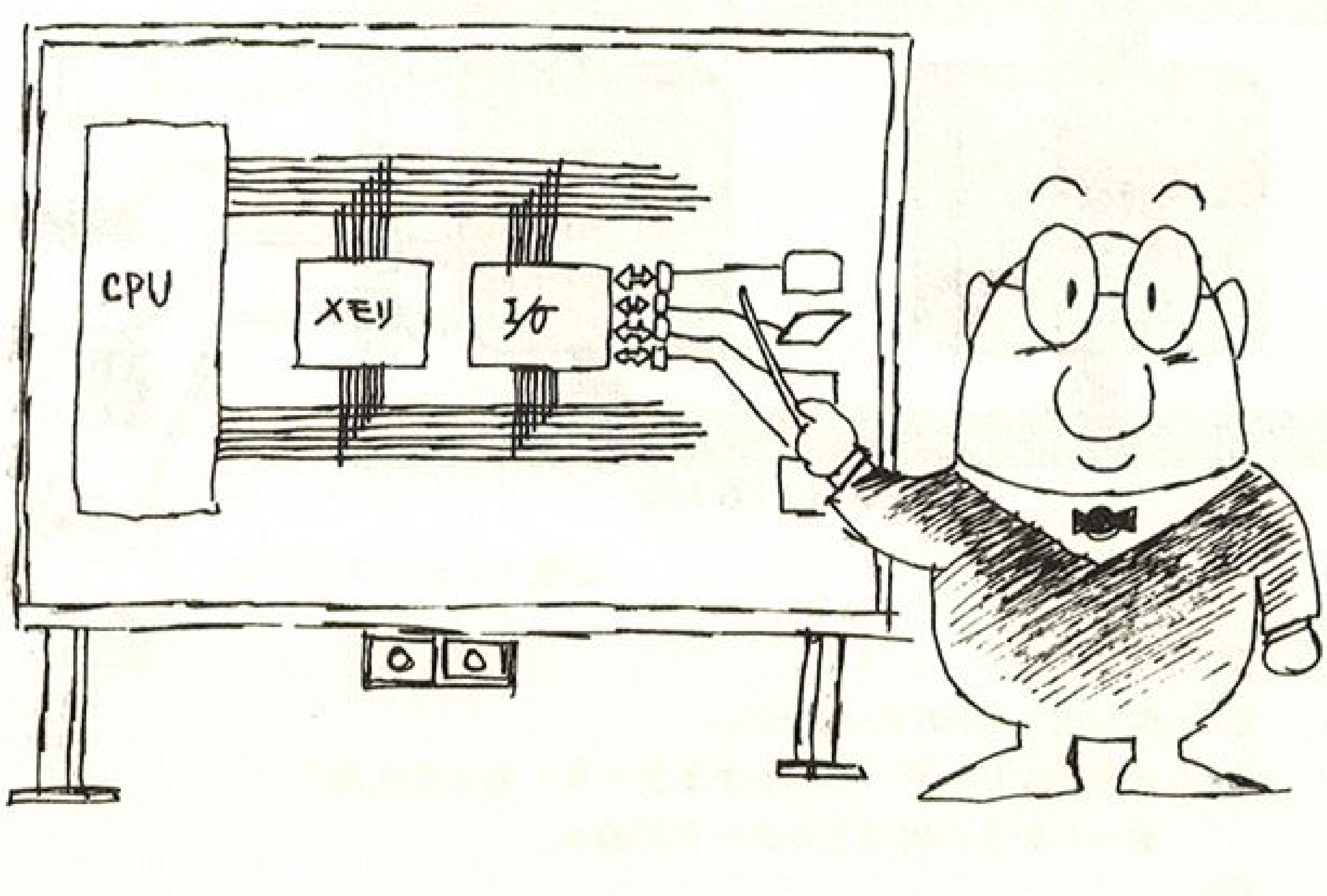


Figure-5.1.1　コンピュータ・システムの構成

それは，本章のタイトルである「CPU」，「メモリ」，「I／O」，「バス」などの基本的なブロックの組合せでしかないのです．そのシステム構成図をFigure-5.1.1に示しました．これはコンピュータの基本です．よく見ておいてください．なおこの図は基本的な構成を示したものであり，ここには描かれていませんが実際にはこれらが互いにうまく動作するように，各種のタイミングをコントロールする部分なども含まれています．この図は，あとの章でもしばしば利用しますのでそのつもりでいてください．



5.2

●CPUの働き

CPUはコンピュータの心臓部であり，Z-80とか8080などのマイクロプロセッサのことをCentral Processing Unit（中央処理装置），略してCPUと呼んでいます．

多くの８ビットのパーソナル・コンピュータに使われているZ-80 CPUは，インテル社の8080 CPUの上位コンパチブルとしてザイログ社が開発したもので，8080のプログラムを完全に実行できるほか，いくつかの命令や機能が追加された強力なCPUとして，世界中のパーソナル・コンピュータで圧倒的に多く採用されているものです．

CPUの内部には，データを一時的に蓄えたり，計算したりするレジスタ群や，プログラムの流れを管理するカウンタや，各種のフラグ用のレジスタ，それにスタックを管理するポインタなどのレジスタがあり，「プログラムの取込み」，「マシン語の解読」，「演算」，「判断」，「分岐」，「入出力」等の正にコンピュータの働きをする中心がこのCPUなのです．CPUの各レジスタやその働きなど詳しくは７章で解説されます．

●● メモリのリード/ライト ●

CPUには16本の「アドレスバス」と呼ばれるアドレス線と，８本の「データバス」と呼ばれるデータ線が接続されています．CPUはメモリをリード／ライト（読出し／書込み）するとき，64Kバイトのメモリの中から任意の１バイトを選択するための16ビットのアドレスデータを，アドレスバス上に出力

します．アドレスバス上にアドレスデータが出力されると，その16ビットの組合せに該当するただ1バイトのメモリが選択されます．選択された1バイトのメモリは，データバスと接続された状態になり（常時は絶縁状態），この間にリード（読出し）であれば，データバス上に出力されているそのメモリ内容をCPUのレジスタに取り込み，ライト（書込み）であればCPUのレジスタにセットされているデータをデータバスに出力して，その1バイトのメモリに書き込ませたりするわけです．このデータのリード／ライトは，いかなる場合もかならず1バイト単位，つまり8ビット単位で行なわれることに注意しておいてください．

アドレスバスとCPU間のデータの流れは，CPU→アドレスバスの一方通行で，CPUはアドレスデータを送り出すのみです．しかし，データバスとCPU間のデータの流れは，CPU↔データバスの双方向であり，CPUは場合によってデータを送り出したり受け取ったりするのです．

これらの様子を次に図解してみましょう．

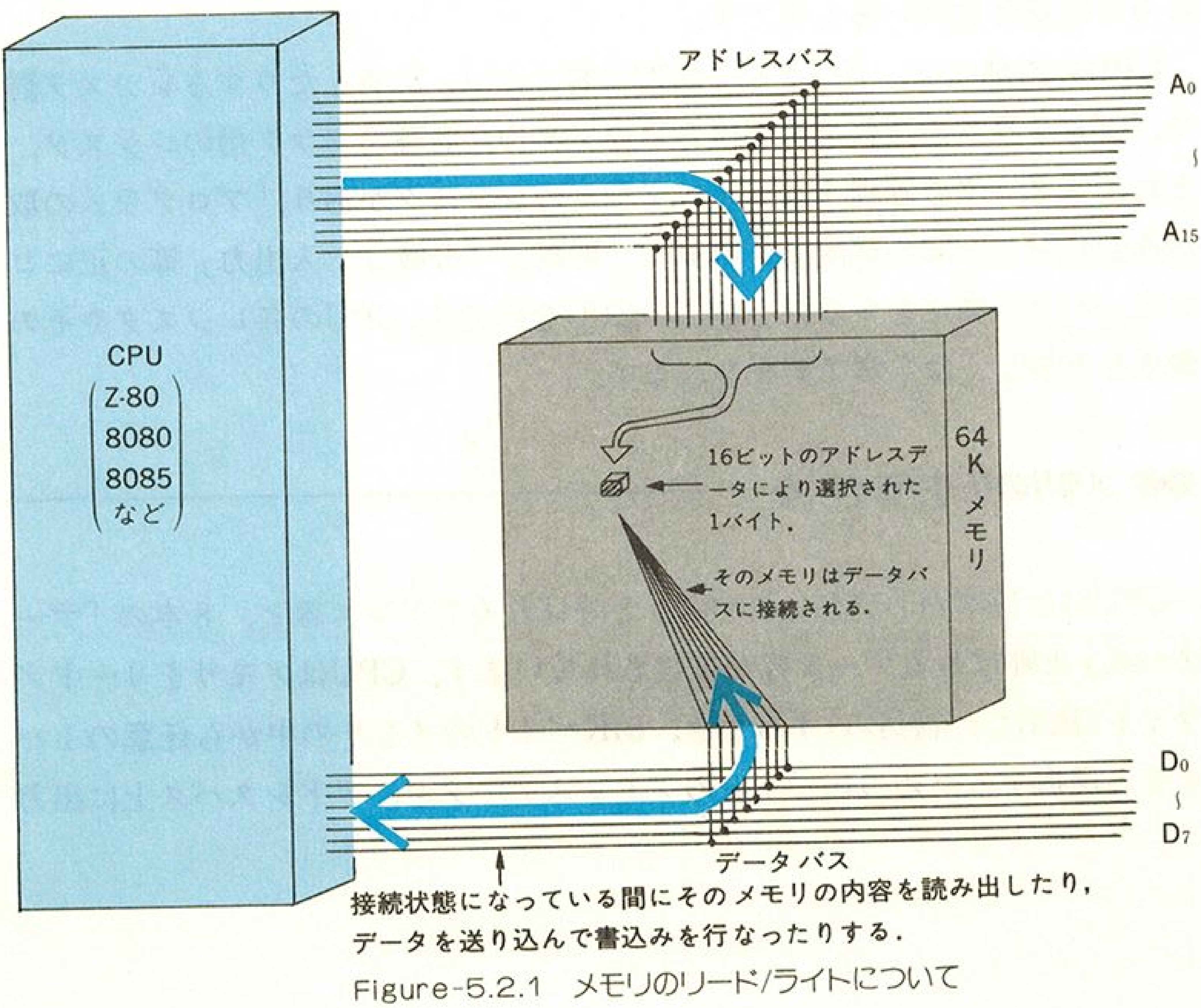


Figure-5.2.1　メモリのリード/ライトについて

5.3

●メモリの種類と構成

Z-80や8080などのCPUは，アドレス線を16本持っており，16ビットで表わすことが可能な，0_HからFFFF$_H$までのメモリ・アドレス空間を持つことができます．つまり，0_H～FFFF$_H$の65536(64Kバイト)個のメモリを直接リード／ライトすることが可能なわけです．具体的なアクセス方法は，前節のCPUの中でふれた通りです．

CPUが直接リード／ライトすることができる64Kバイトのアドレス空間にあるメモリを，「メインメモリ」と呼んで，バンク切換え(後述)用の64Kバイト以外のメモリや，ディスク装置等の外部メモリと区別しています．

●● ROMとRAM ●

メインメモリには，自由に書込みのできないROM(Read Only Memory)と，書込み読出しが自由にできるRAM(Random Access Memory)とがあることはすでにご承知のことでしょう．

電源を入れると組込みのBASIC言語が起動するパーソナル・コンピュータでは，4章でも解説しているBASICインタープリタのマシン語プログラムがメインメモリのROMに固定されているわけです．ROMの内容はCPUが書換えを行なうことはできません．またコンピュータの電源を切っても消えてしまうことはなく，実用上永久に保存されます．一方，ユーザー・プログラムや各種データエリア，それにコンピュータが働くために必要とするワーキング



エリア等は，自由にリード／ライトを行なわなければなりませんので，当然のことながらRAMエリアとなっています．

RAMもROMもCPUから見れば，アドレスバス，データバスにつながる同じメモリであり，実行するプログラム上で両者の取扱い上の差異はありません．ただROMは書込みができないというだけのことです．

参考までにROMの種類は大きく分けてふたつあり，それらは「マスクROM」および「PROM」と呼ばれています．マスクROMは，ROMを製造する時すでに構造的にプログラムが(書き込まれると言うよりは)刻み込まれており，大量生産が可能です．そのかわりどんな手段を使っても書き換えることはできません．もうひとつのPROMはProgramable ROMという意味であり，専用のPROM書き込み器で自由に書込みが可能です．また書き込まれているプログラムは，専用の消去器で消去することが可能で，消去されたPROMは，再び新しいプログラムを書き込むことができます．

●● バンク切換え

いろいろなアプリケーション・プログラムを考えていくと，マシンに組込みのBASIC言語を使わない場合BASICのROMはじゃまとなり，そこをRAMエリアとして使いたい場合があります．また，同じエリアをあるときはROM，あるときはRAMとして使いたいとか，64Kバイトよりもっと大きなメモリエリアを使いたい，などのいろいろな要望が出てきます．それらを解決する手段として「バンク切換え方式」があり，多くのパーソナル・コンピュータで採用されています．

そのバンク切換え方式の原理を次に図解しておきましょう．

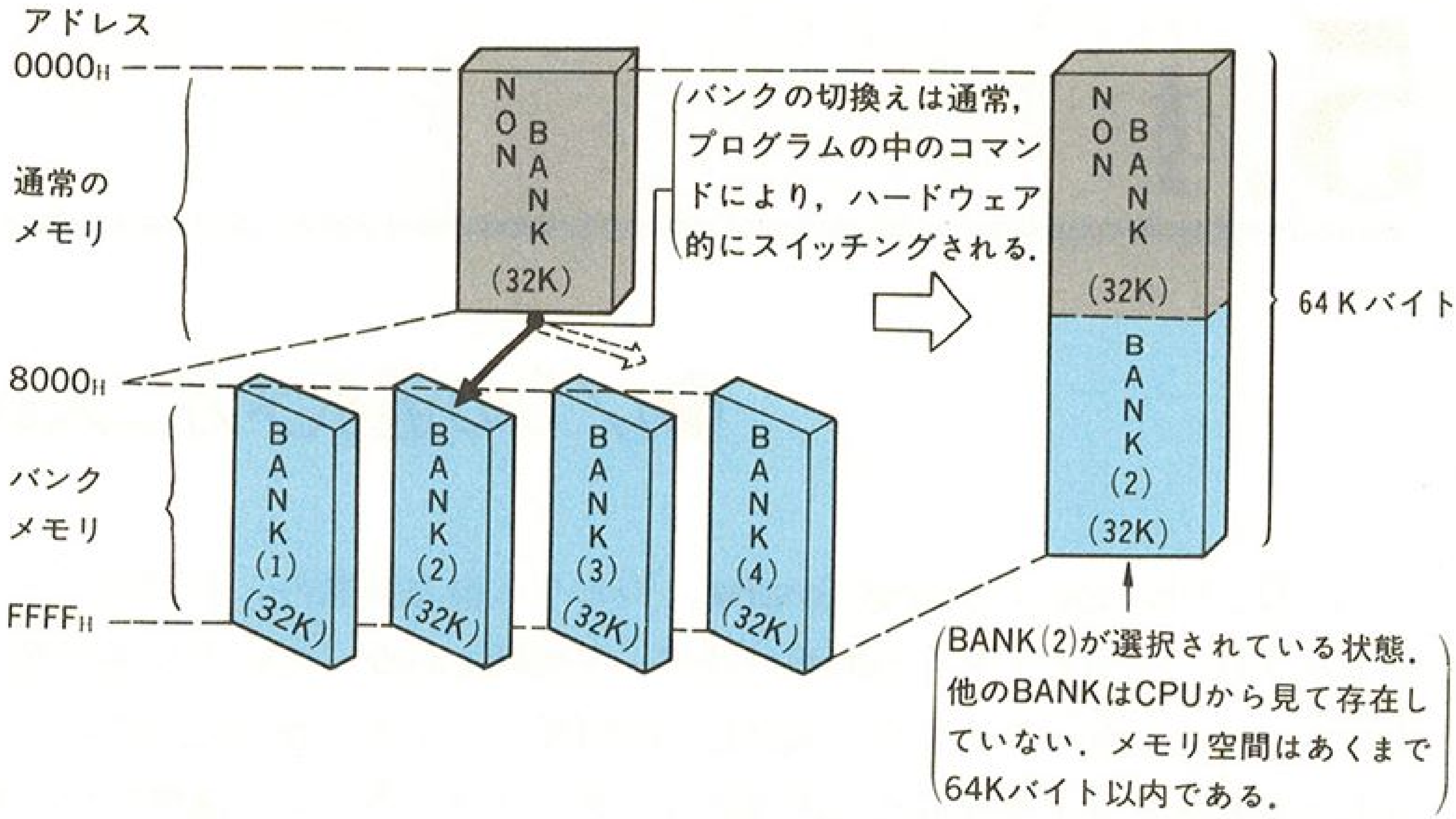


Figure-5.3.1　上位の32Kバイトが4バンク切換えになっている160Kバイトのメモリ構成

この図の例では同一のメモリエリア（アドレス8000$_H$～FFFF$_H$の32Kバイト）に4つのバンクが同居しています．この4つのバンクのうちのひとつを，場合場合によって選択し，一連のメモリ空間として使用するわけです．CPUから見ると，同一メモリエリアにある複数のバンクは，ある時点ではただひとつのみが生きており，他のバンクはまったく存在していません．バンク切換え方式で，全メモリ容量が64Kバイト以上であっても，ある時点で機能するメモリは，かならず64Kバイトのアドレス空間ですので，ここのところを誤解しないようにしてください．

バンクを切り換える具体的な方法としては，ソフトウェア・コントロール，つまり実行するプログラムの中でバンク切換えを行なう方式が多くのパーソナル・コンピュータで採られています．これはプログラムの中でコマンドを実行することにより，切換え部分のハードウェア的なスイッチが動作し，切換えが行なわれるという方式です．

5.4

●I／Oの選択メカニズム

「I／O」とはInput／Output System，つまり入出力装置のことです．コンピュータは，CPUとメモリの間だけでデータ処理を行なっていても何の役にも立ちません．キーボードから入力し，CRTやプリンタに出力し，ディスクなどの外部メモリとやり取りし，他のコンピュータとラインで通信したりするなど，周辺装置が接続されることにより私たちはコンピュータの力を利用することができるのです．また，機械をコントロールするだけのコンピュータ（コンピュータを利用したコントローラと言った方が適切）では，機械からいろいろな状態信号（ステータス情報と言う）を受け取り，それによって機械をコントロールするための様々な信号を出力しなければなりません．

様々なデータや信号をやり取りする，つまり入出力するということは，具体的にはFigure-5.1.1のデータバスに，周辺装置のデータや信号を接続するということになります．その接点にあたる部分がI／Oポート，すなわち入出力ポートです．Portは文字通り「港」です．それぞれのポートを介して「8ビットのデータ」や，これをスムーズにやりとりするために，コンピュータと相手の機器との状態を知らせあう「ステータス信号」などを受け渡し（入出力）する場所です．

ポートの数は通常最大で256ポートまで設けることが可能です．つまりFigure-5.1.1のアドレスバスに接続されている8本のアドレス線で表わせる00_H〜FF_Hのポート・アドレスによりそれぞれのポートを選択します．

I／Oの選択メカニズムは，Figure-5.2.1でも示されているメモリの選択と

同様です．メモリの場合は，16ビットのアドレス線をフルに使って，0000_H ～$FFFF_H$の65536個のメモリ・アドレス空間の内のひとつを選択しますが，I／Oポートには８本(通常は下位の８ビット)のアドレスバスが接続されているだけなので，I／Oアドレス空間はぐっと小さくなり，８ビットで表わせる00_H～FF_Hの256個のポートの内のひとつを選択できるだけです．

メモリをアクセスする場合もI／Oをアクセスする場合も同じアドレスバスを使いますが，メモリのリード／ライトの場合は16ビットのアドレスデータが，またI／Oポートの入出力の場合は８ビットのアドレスデータがCPUから出力されます．同じアドレスバスを共有して使っていますが，入出力とメモリ操作のときとを区別するためのコントロール信号が別に出力されますので，問題は起こりません．

I／Oアドレスによるポートの選択と，入出力データの関係などを次に図示しておきます．

なお，これらの入出力に関する具体的な実習は，8.11，9.3，9.4の各節で，マシン内臓のブザーと，プリンタ用インターフェイスを使って行なっています．

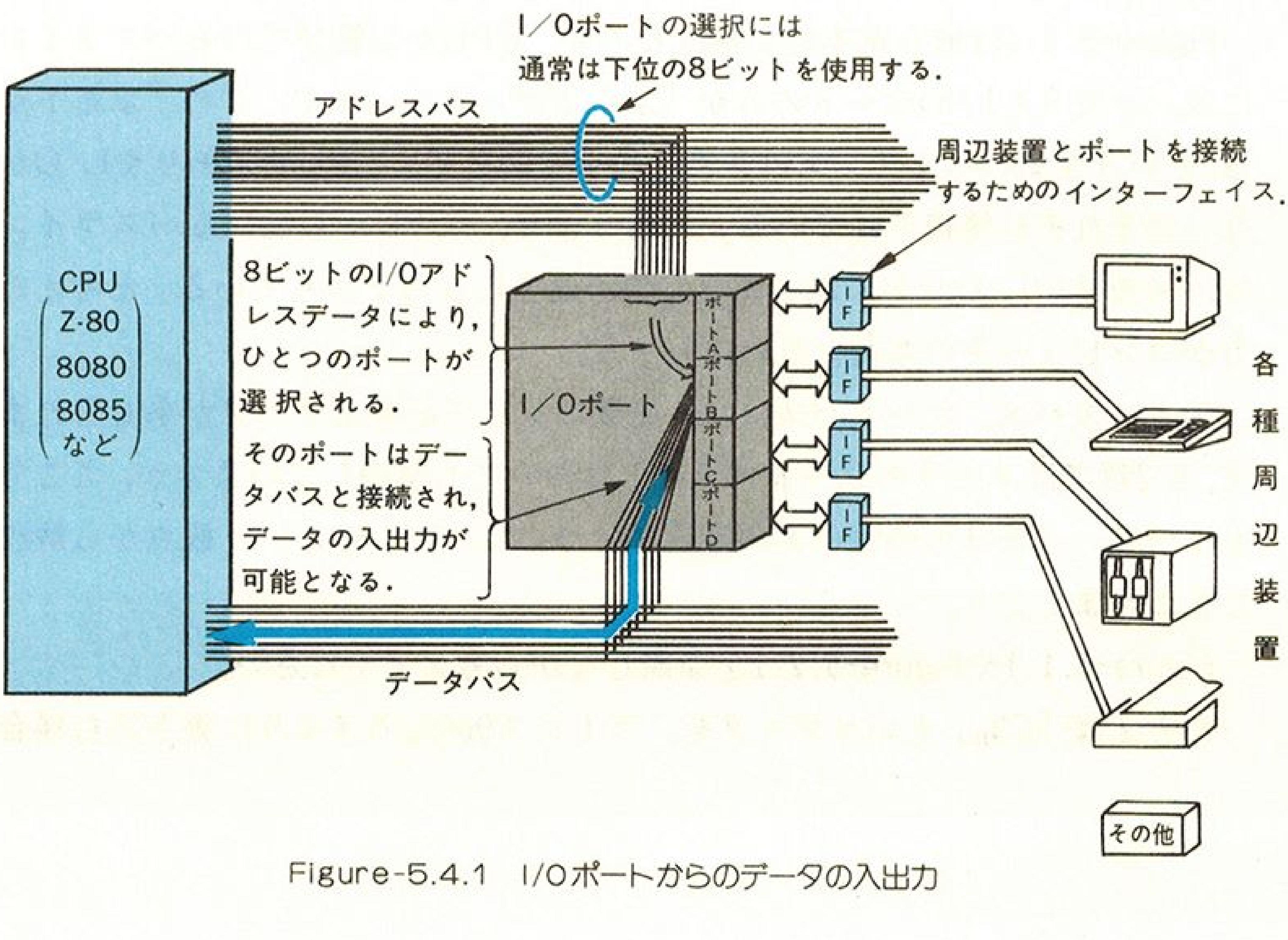


Figure-5.4.1　I/Oポートからのデータの入出力

5.5

●バスの概念

コンピュータの内部では，各種のデータのやり取りのために，「バス」という考え方が採られています．バス＝Busは「乗物」というほどの意味なのでしょう．データが乗るハイウェイ，つまりデータが流れる幹線というイメージを持てばよいでしょう．

基本的なバスには，「アドレスバス」と「データバス」のふたつのバスラインがあります．Figure-5.1.1に図示されているように，バスラインはCPUから延びています．言い替えれば，CPUにはバスラインがつながっているだけと考えてもよいでしょう．

Figure-5.1.1は最も基本的な構成なので，CPUから延びているバスラインには，メモリとI／Oポートのみが「ぶら下がって」います．この「ぶら下がる」という言葉のニュアンスは非常に大切なことで，「CPUやメモリやI／Oが互いにそれぞれ接続されている」と言うより，「CPUから延びるバスラインに，メモリやI／Oやその他必要なものが適当にぶら下がっている」と考えた方がコンピュータの本質に近いでしょう．

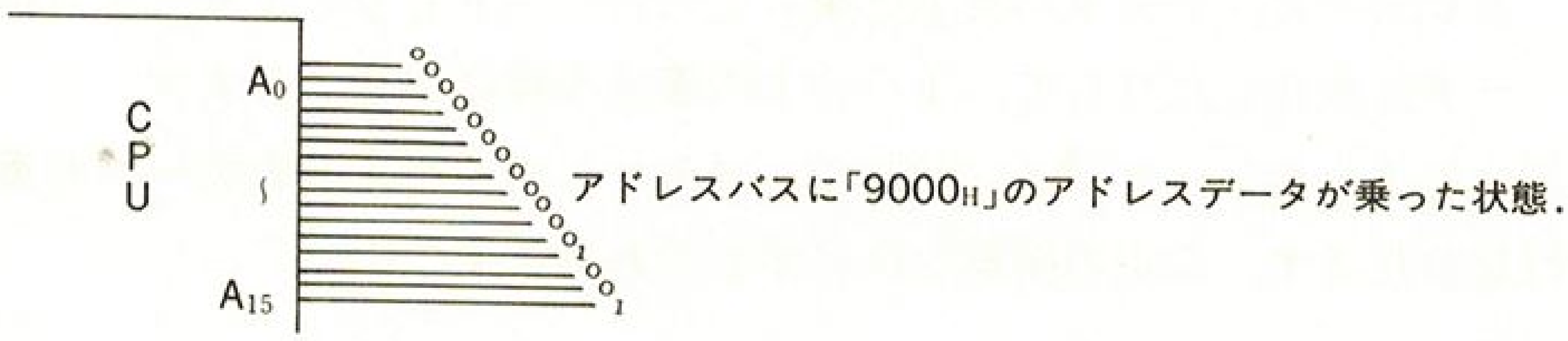
アドレスバス，データバスには，必要に応じて必要なデータが乗せられます．5.2章ではメモリのリード／ライトについても解説していますが，ここでもう一度，メモリへのデータの書込みについて，「バス」という観点から解説しておきましょう．

Figure-5.1.1やFigure-5.2.1を参照しながら考えてください．

たとえば「$C3_H$」というデータを，アドレス9000_Hのメモリに書き込む場合

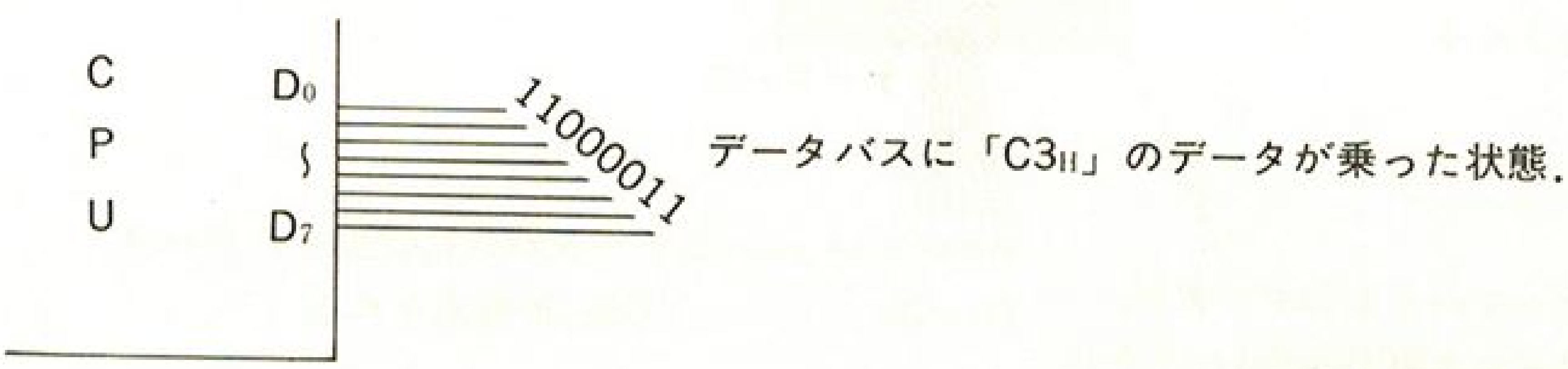
を考えましょう．事は次のように運びます．

①アドレスバスに9000_HというアドレスデータがCPUから送り出されます．具体的にはA15, A14, A13, …… A0の16本のアドレスバスに，1001000000000000というデータが乗ることになります．



②アドレスデータが出力されると，64Kバイト(65536バイト)のメモリ・アドレス空間にあるただ1バイトのメモリのみが選択されます．ここではアドレス9000_Hが選択されました．

③アドレスデータによる1バイトのメモリの選択が完全に行なわれた頃合をみて，CPUのレジスタにセットされていた「$C3_H$」というデータ(11000011)が，データバスに送り出されます．



④データバスに送り出した「$C3_H$」のデータが完全に安定した頃合をみて，CPUから全メモリ（64Kバイトの全メモリ）に対して同時に「WRITE（書き込み）」信号を出します．WRITE信号はFigure-5.1.1やFigure-5.2.1には描かれていませんが，コンピュータ・システムの働きをコントロールする数本のコントロール信号のひとつです．

⑤WRITE信号はすべてのメモリに対して送り出されますが，アドレスデータにより選択されているただ1バイトのメモリを除き，他のメモリに対しては無効です．実際の書込みが行なわれるのは選択された1バイトだけです．

この例ではアドレス9000$_H$が選択されているので，この1バイトのメモリだけはCPUからの「WRITE」信号を受け取ると，直ちにデータバス上のデータ「C3$_H$」を記憶します．

⑥完全に記憶が行なわれた頃合をみて，「WRITE」信号はOFFとなり，アドレスバス，データバス上に乗っていたデータもなくなり，また他のデータに変化したりして，1バイトの書込み動作は完了します．

以上のようなシーケンス(手順)で，メモリへの1バイトのデータの書込みが行なわれます．以上の図解を次に示しておきます．

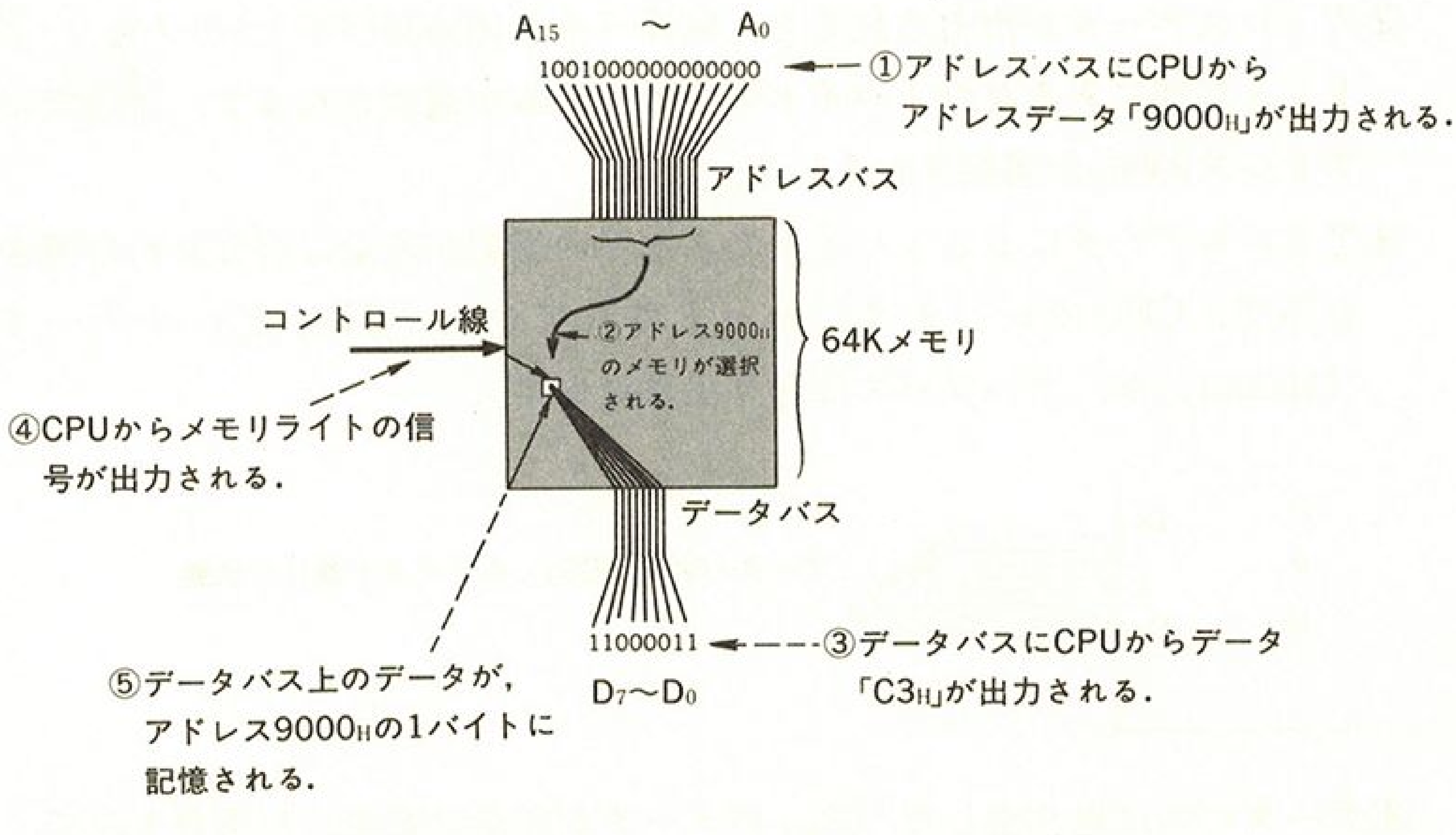


Figure-5.5.1　メモリ ライトのシーケンス

「バス」の概念はだいたいつかめたでしょうか．アドレスバス，データバスに，CPUやメモリやI/Oがぶら下がり，あるものはバスにデータを出力し，またあるものはバス上のデータを取り込み，次の時点では同じバス上にまったく別のデータが乗っている，というように，バスはコンピュータ内部のデータの運搬路として非常に重要な働きをしているのです．

6

ニーモニックとアセンブラ

さていよいよマシン語の「記号」や「言葉」の登場です．みなさんはすでにBASIC言語を使っていることと思いますが，BASICの言葉の使い方がわかるなら，マシン語の言葉は大丈夫です．マシン語の一つひとつは非常にシンプルですから，まずはやってみてください．

本章では，マシン語の記号である「ニーモニック」と，それを使ってプログラミングを行なう言葉である「アセンブラ」について解説します．特にハンド・アセンブルについては詳しく説明しています．章の終わりには本格的なCP/Mベースのアセンブラの実行例を紹介していますので，これは将来の参考にしてください．

6.1

●ニーモニックとアセンブリ言語

4.2章で実行したFigure-4.2.2のマシン語プログラムをもう一度ここに示します.

```
21 00 00 11 00 A0 01 64 00 ED B0 C9
```

Figure-6.1.1　短いマシン語のプログラムの一例

これはわずか12バイトですが，ある動作をするひとつのプログラムです．しかしこの12バイトの16進数の羅列を眺めていても，どういう内容なのかよくわかりませんね．このようにマシン語の16進数が並んでいるだけでは，よほどマシン語に精通している人でない限り，その内容を理解することはできません．もちろんプログラムですから，ある規則に従った並びをしていますが，たとえ精通している人でも，長いプログラムであれば，その全体を的確に見通すことは不可能でしょう．

そこでマシン語と一対一に対応した人間にわかりやすい「言葉」が必要となるわけで，マシン語の「ニーモニック」を使った「アセンブリ言語」が広く使われているのです．

さきほどの12バイトのマシン語プログラムは，4．2章でも解説していますが，アドレス0000_Hからの100バイトのメモリ内容を，アドレス$A000_H$からの100バイトのメモリにそっくりコピーする（ブロック転送と言う）プログラムです．

この12バイトのマシン語をアセンブリ言語で表現してみましょう．

```
LD      HL,0000H
LD      DE,0A000H
LD      BC,100
LDIR
RET
```

Figure-6.1.2　12バイトのマシン語をアセンブリ言語で表現したもの

Z-80のアセンブリ言語ではこのように記述し，この形式を「アセンブリ・ソース・プログラム」と言います．「アセンブリ言語で書いた，マシン語の素(Source)になるプログラム」という意味です．

「アセンブリ言語」と「ニーモニック」の関係はというと，マシン語の命令の一つひとつを文字で表わしたのがニーモニックであり，そのニーモニックを使ってプログラミング言語の体系にまとめたものがアセンブリ言語であると理解しておいてください．

このようにアセンブリ言語で書くと，16進数のマシン語よりはだいぶ言葉らしくなりましたね．この言葉が何を言っているのか，もう少し詳しく解説しましょう．

アセンブリ・ソース・プログラム（アセンブリ言語で書いたプログラム）	ニーモニック：オペコード（主命令）	ニーモニック：オペランド（主命令が対象とする相手，条件，パラメータ）	オブジェクト・プログラム（マシン語）
ORG 9000H			
LD HL,0000H	LD（ロードせよ）	HL ,0000H（CPUのHとLレジスタに）（値0000を）	21 00 00（ロード・アドレスの進む順序）
LD DE,0A000H	LD（ロードせよ）	DE ,0A000H（CPUのDとEレジスタに）（16進の値A000を）	11 00 A0
LD BC,100	LD（ロードせよ）	BC ,100（CPUのBとCレジスタに）（10進の値100を）	01 64 00
LDIR	LDIR（ブロック転送せよ＊）		ED B0
RET	RET（サブルーチンを終わり，メインルーチンに戻れ.）		C9

Figure-6.1.3　アセンブリ・ソース・プログラム，ニーモニック，マシン語の関係

＊HLレジスタペアの値で示されるアドレスから，BCレジスタペアの値で示されるバイト数分のメモリ内容を，DEレジスタペアで示されるアドレスへコピーせよという命令．

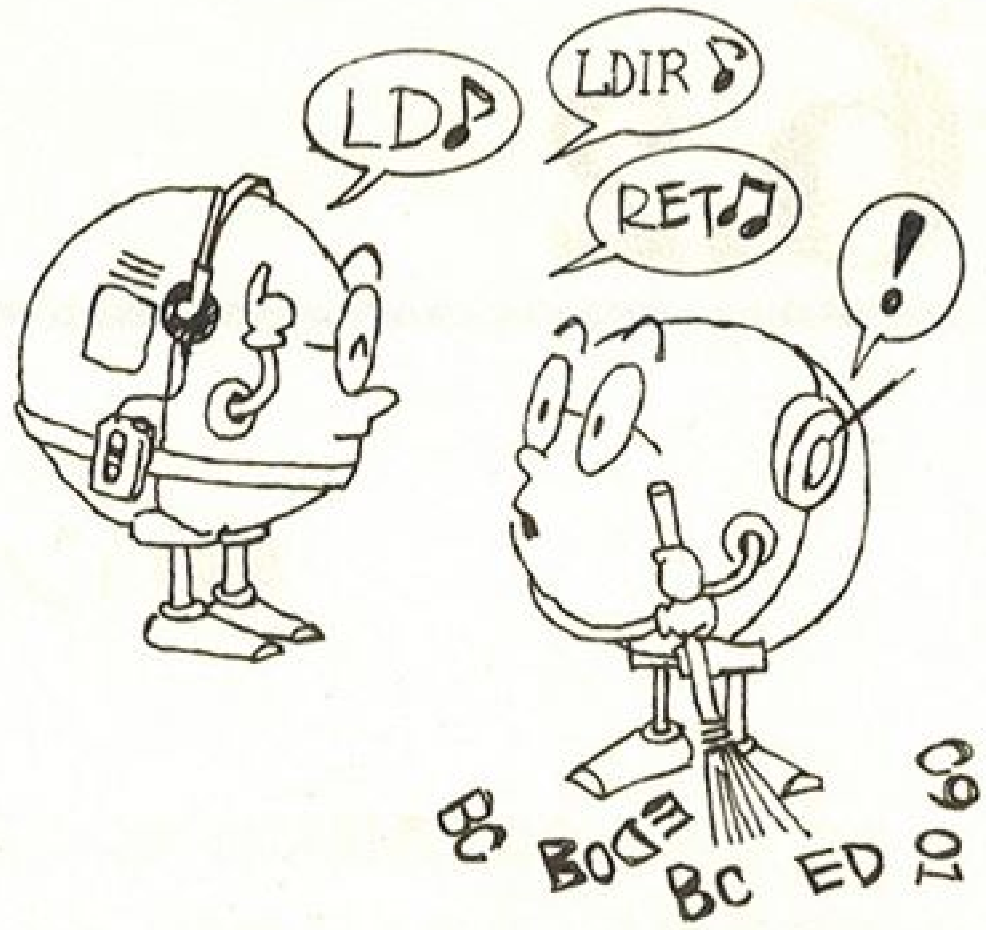


Figure-6.1.3を見てどう感じますか．意外と単純明快だとは思いませんか．このようにマシン語の一つひとつはたいへんシンプルなのです．

ここでは，「LD」，「LDIR」，「RET」の3つのニーモニックのみを使った非常に簡単なプログラムを使って，マシン語とニーモニックならびにアセンブリ言語の関係を示しています．マシン語のプログラムを作成するには，頭から16進数のマシン語コードをコツコツと書いていくのではなく，まずアセンブリ・ソース・プログラムを書かなければならないことが理解されたと思います．

最終的にコンピュータが実行可能なプログラムは，16進数で表わされたマシン語コード（私たちが単にマシン語と言っているもの）であり，これを「オブジェクト・プログラム」と呼んでいます．Figure-6.1.3の右側の部分にあたる16進数がそれです．

整理すると，マシン語のプログラムを作成する手順は，次のようになります．

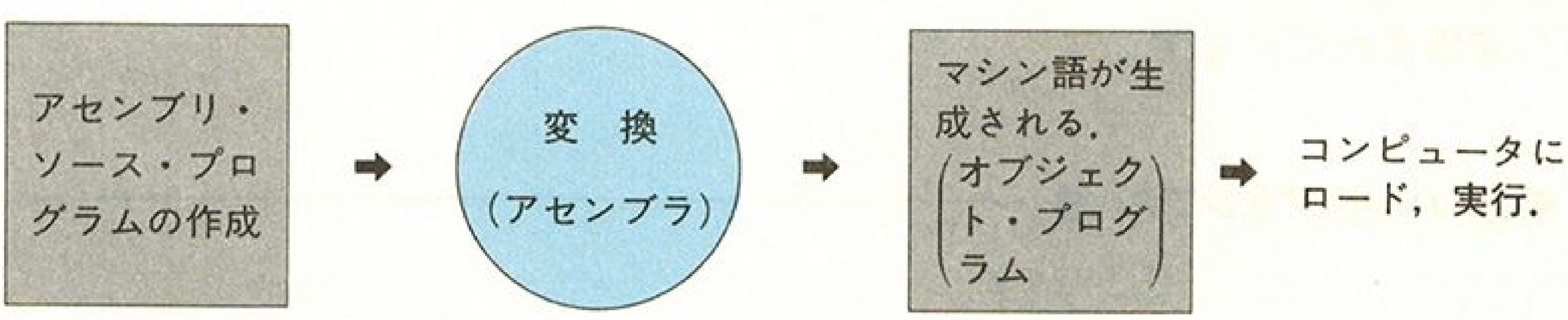


Figure-6.1.4　マシン語プログラム作成過程

「変換」の部分については次の節で解説しますが，これが「アセンブラ」と呼ばれるものなのです．

6.2

●ハンド・アセンブルとアセンブラ

コンピュータは，直接にはマシン語のプログラムしか実行できないことは，今まで何回もくり返して解説しました．これと同様に，アセンブリ・ソース・プログラムをコンピュータにロードしても，コンピュータはまったく働きません．

コンピュータが実行可能なマシン語のプログラムを作成するには，まずアセンブリ・ソース・プログラムを作成し，そこからオブジェクト・プログラムを導き出すわけです．それが前節のFigure-6.1.4の「変換」にあたる「アセンブラ」と呼ばれる道具なのです．

Assemblerとは，つまりアセンブルするものという意味で，具体的にはソフトウェア製品であり，一般的に広く使われている例としては，デジタルリサーチ社の「CP／M ASM」や「MAC」，マイクロソフト社の「MACRO-80」などのCP／M上で使用するアセンブラが，本格的な実用アセンブラとして多く使用されています．

●● ハンド・アセンブル

アセンブリ・ソース・プログラムをアセンブルするには，ソフトウェア・ツール(道具としてのソフトウェア)が必要ですが，Figure-6.1.3(アセンブリ・ソース・プログラム，ニーモニック，マシン語の関係)のような簡単なプログラムなら，アセンブラを使わずに，巻末付録にある「機能別主要命令表」を見て，

手でアセンブルすることが可能です．このように紙とエンピツと手で行なうやり方を俗称(今や正式用語？)「ハンド･アセンブル」と呼んでいます．

ここで前節のアセンブリ･ソース･プログラムを例に，ハンド･アセンブルのやり方を示しておきましょう．次の図解をもとに，解説を行ないます．

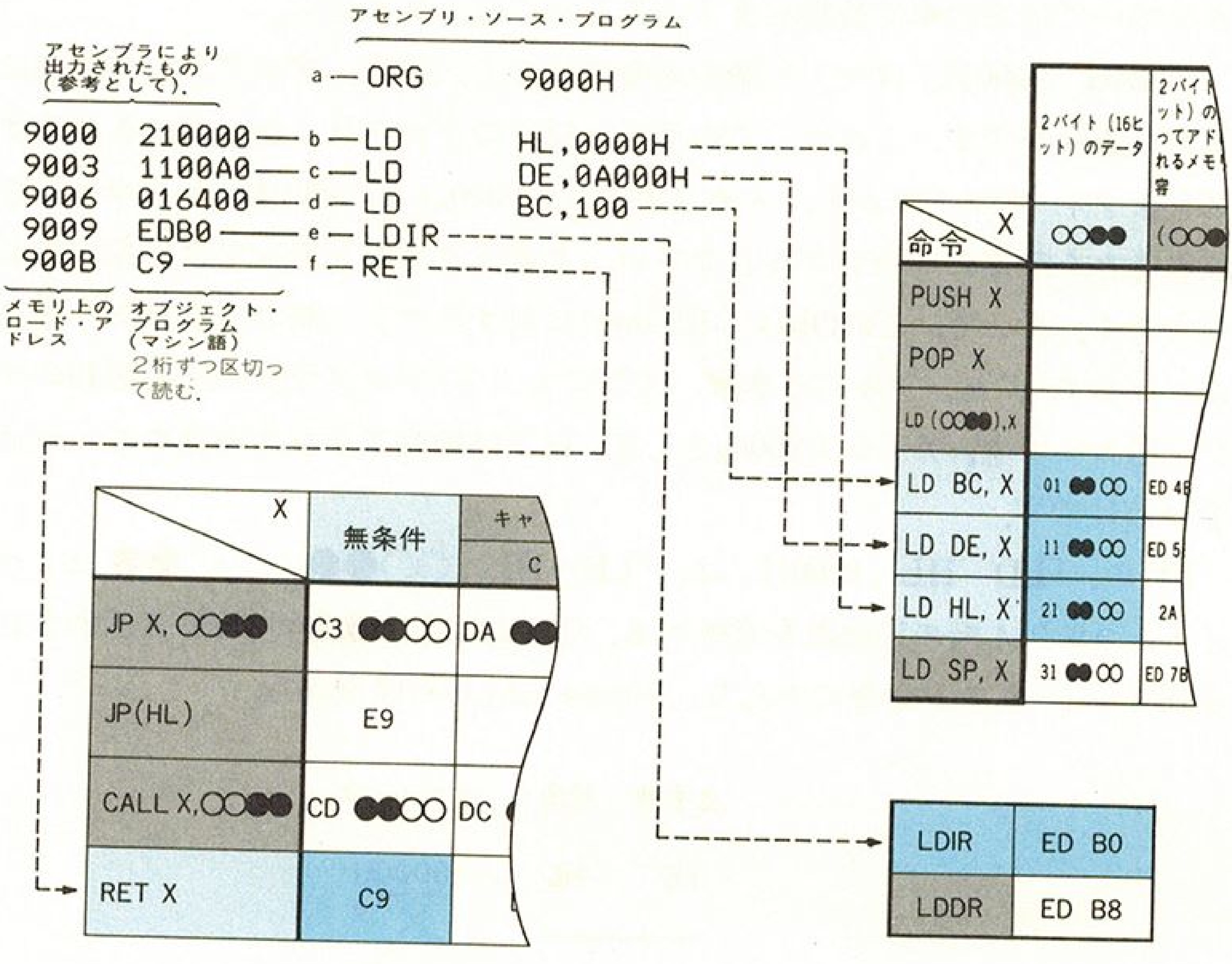


Figure-6.2.1 ハンド･アセンブルの方法

ハンド･アセンブルの方法を上図のアセンブリ･ソース･プログラムのa～fの各ラインについて説明していきましょう．それぞれのマシン語の働きについては次の章「コンピュータの中心CPU」で解説しますので，ここではハンド･アセンブルについてのみ解説します．ここで単に「アセンブラ」と表現しているのはソフトウェア･ツールのアセンブラを指していますので，ハン

ド・アセンブルと区別してください.

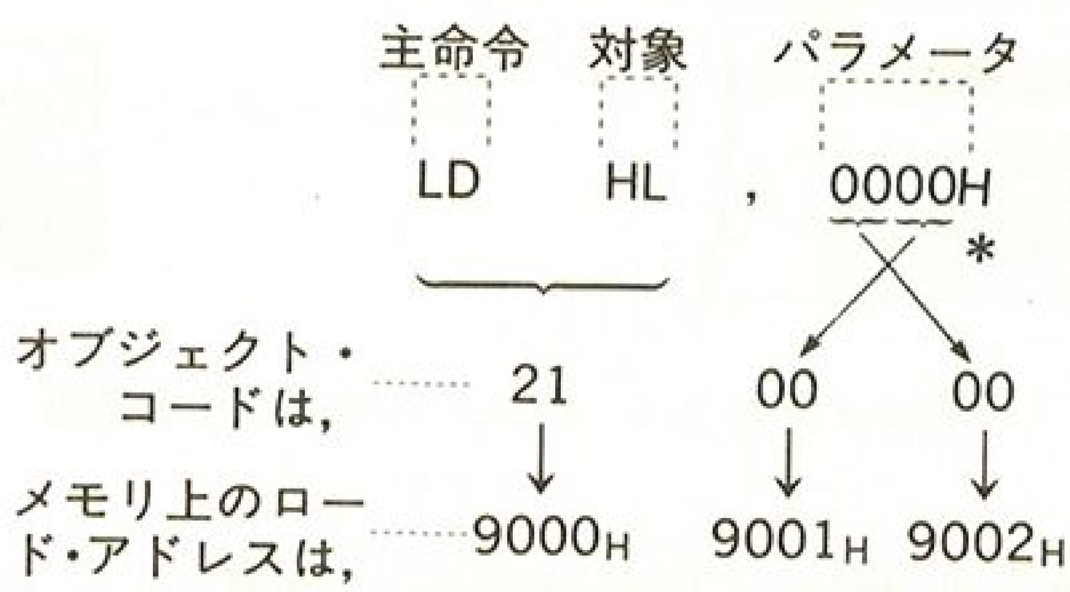
ではまずa行の

ORG 9000H

からハンド・アセンブルの方法について説明していきましょう.マシン語の働きについては次の章で解説しますのでここではふれません.

「ORG　9000H」はマシン語の命令ではなく，通常，プログラムの冒頭に書かれる命令です．これは，このライン以下のプログラムから生成されるオブジェクト・プログラムを，メモリアドレス9000_H(この例では)からロードされることを前提にアセンブルしなさい，と言うことをアセンブラに指示する命令です．よって，このORG(ORiGine（オリジン）)に対するマシン語コードはありません．ハンド・アセンブルのときは，オブジェクト・プログラムの一番最初のマシン語コードを，アドレス9000_Hとして，以下は順次アドレスを進めていけばよいわけです.

b行の,「LD　HL ,0000H」は,「LD　HL ,○○●●」(○○●●は2バイト，つまり4桁の16進数を意味する．たとえば○○●●＝9050)で表わされる16ビット・ロード命令にあたり，Figure-6.2.1の対照表から,



となります.

c行の,「LD　DE ,0A000H」も同じグループの「LD　DE ,○○●●」にあたり，対照表から,**

*4桁の16進数（つまり16ビットのパラメータ）をアセンブルする場合，メモリ上では上位と下位の8ビットを逆にする（○○●●→●●○○）ので注意すること.

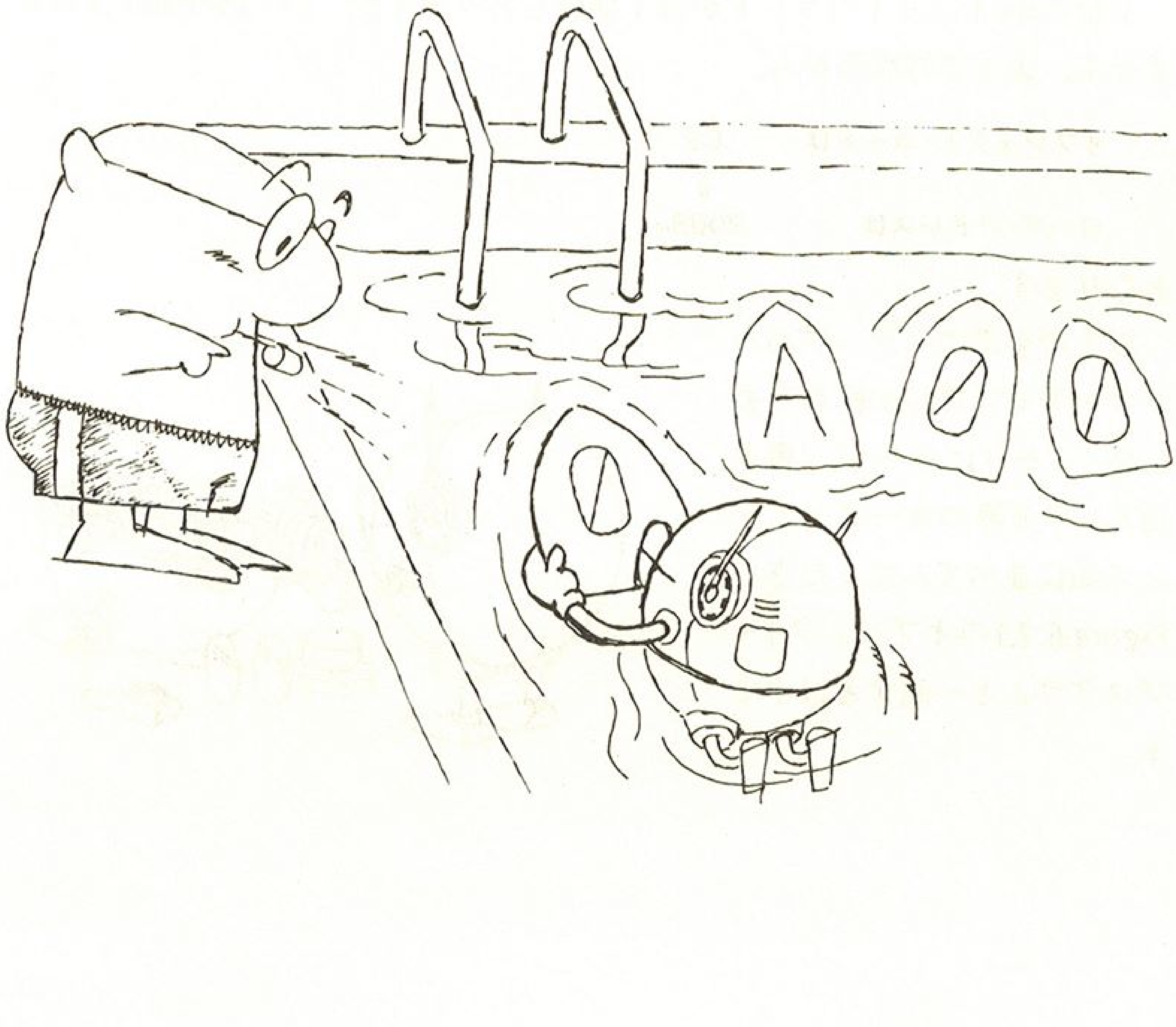
**本書では対照表としてAPPENDIX 3に機能別主要命令表を用意してある．8章以後でハンド・アセンブルする場合も参照すること.

```
                                   LD  DE,      0A000H
                                     ↓
オブジェクト・コードは,……………           11          00     A0
                                     ↓           ↓      ↓
メモリ上のロード・アドレ………………     9003H       9004H  9005H
スは続きから
```

となります.

なお,「0A000H」の最初の0は,アセンブラが16進数の値「A000」のAを,アルファベットのAではなく,16進数のAとして正しく解釈できるように,前に置く約束事になっているための0です. A,B,C,D,E,Fで始まる16進数の頭には,かならず0を置きます. ハンド・アセンブルを行なう場合は,人間には前後の関係で数値であることが判別できますので,特に必要はないのですが,習慣として身に付けておいてください.

d行の,「LD　BC ,100」も同じグループの「LD　BC ,○○●●」にあたり,対照表から,



LD　BC,　100 ……… Hが付いていないので10進の100である.

↓ ……… 10進数の100を4桁の16進数で表わすと0064_H.

0064_H

オブジェクト・コードは, ……… 01　64　00

メモリ上のロード・アドレスは続きから ……… 9006_H　9007_H　9008_H

となります.

e行のLDIRにはオペランド(主命令が対象とする相手, 条件, パラメータ. Figure-6.1.3の表を参照)はありません. これはブロック転送命令グループのひとつであり, 対照表から,

オブジェクト・コードは ……… ED　B0

ロード・アドレスは ……… 9009_H　$900A_H$

の2バイトとなります.

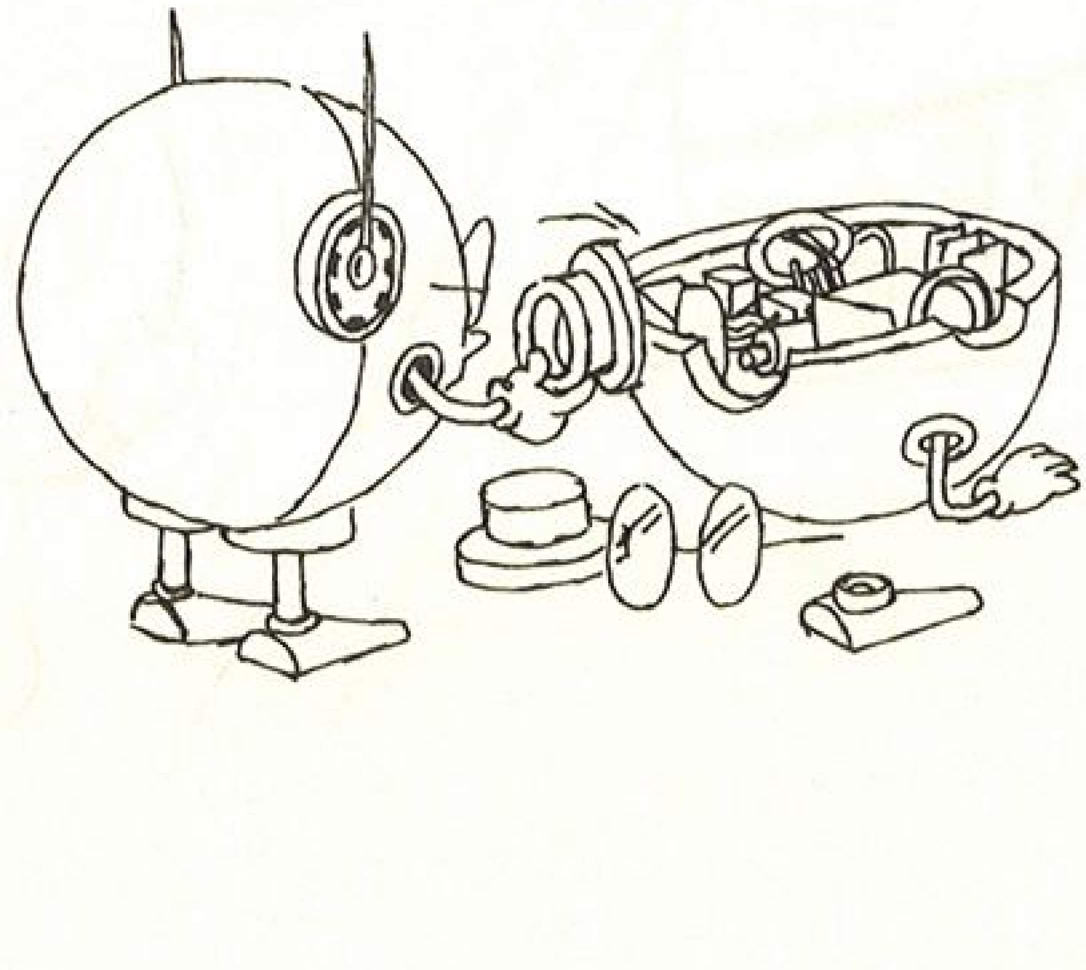
f行のRETにはオペランドが付く場合もありますが, この例の場合はありません. よって対照表から,

オブジェクト・コードは ……… C9

ロード・アドレスは ……… $900B_H$

となります.

以上で全部のステップのハンド・アセンブルが終わりました. メモリにロード(書き込み)する際のロード・アドレス順に並べてみてください. Figure-6.2.1のオブジェクト・プログラムと一致するはずです.



●● 本格的なアセンブラの例 ●

さきほどはハンド・アセンブルの方法を解説し，アセンブリ・ソース・プログラムとマシンコードの関係などを学習しました．しかしハンド・アセンブルは能率が悪く誤りも多いので，このような学習用もしくは，ごく短いソース・プログラムに使用する程度で，通常のソフトウェア開発においては，ソフトウェア・ツールのアセンブラを使用します．

Z-80や8080系のパーソナル・コンピュータで，本格的なアセンブラを使用するには，何と言っても「CP／M」というオペレーティング・システム(略してOSと言う)を使うことです．これがソフトウェア開発や種々のソフトを利用するための最も優れた環境が得られる最善の方法です．

CP／M上の本格的アセンブラについては，本節の冒頭にも，「MAC」と「MACRO-80」などの製品名を紹介しましたが，ここではクロメンコ社のアセンブラを実際に使ってみます．このアセンブラは，Z-80専用のもので(MAC, MACRO-80はZ-80，8080の両方がアセンブル可能)，筆者が場合場合によって使い分けているいくつかのアセンブラのひとつです*．CP／Mの環境の良さの一例として，このようにいくつものアセンブラの中からその時の仕事に適したものを自由に選択できることもあるのです．

アセンブラによるマシン語ソフトウェアの本格的な開発には，最低でも次の3つの基本的なソフトウェア・ツールが必要です．これを次の図で示します．

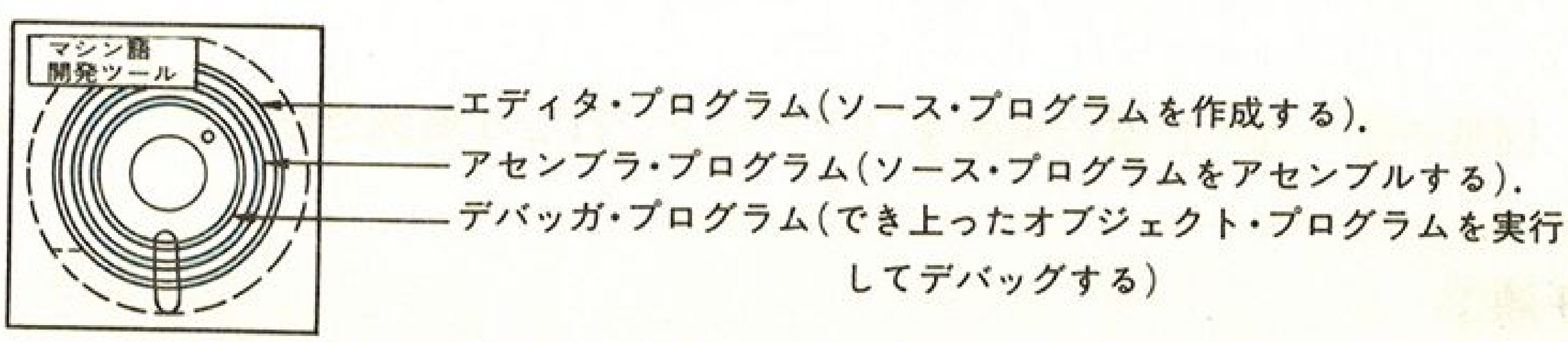


Figure-6.2.2 アセンブラによるソフトウェア開発に必要な3つの基本ツール

*本来はCDOSという準CP／M的なOS上のものだが，筆者が一部変更してCP／M用にしたもの．

次に，ソフトウェア開発の手順を，アセンブルを中心に図で示しましょう．

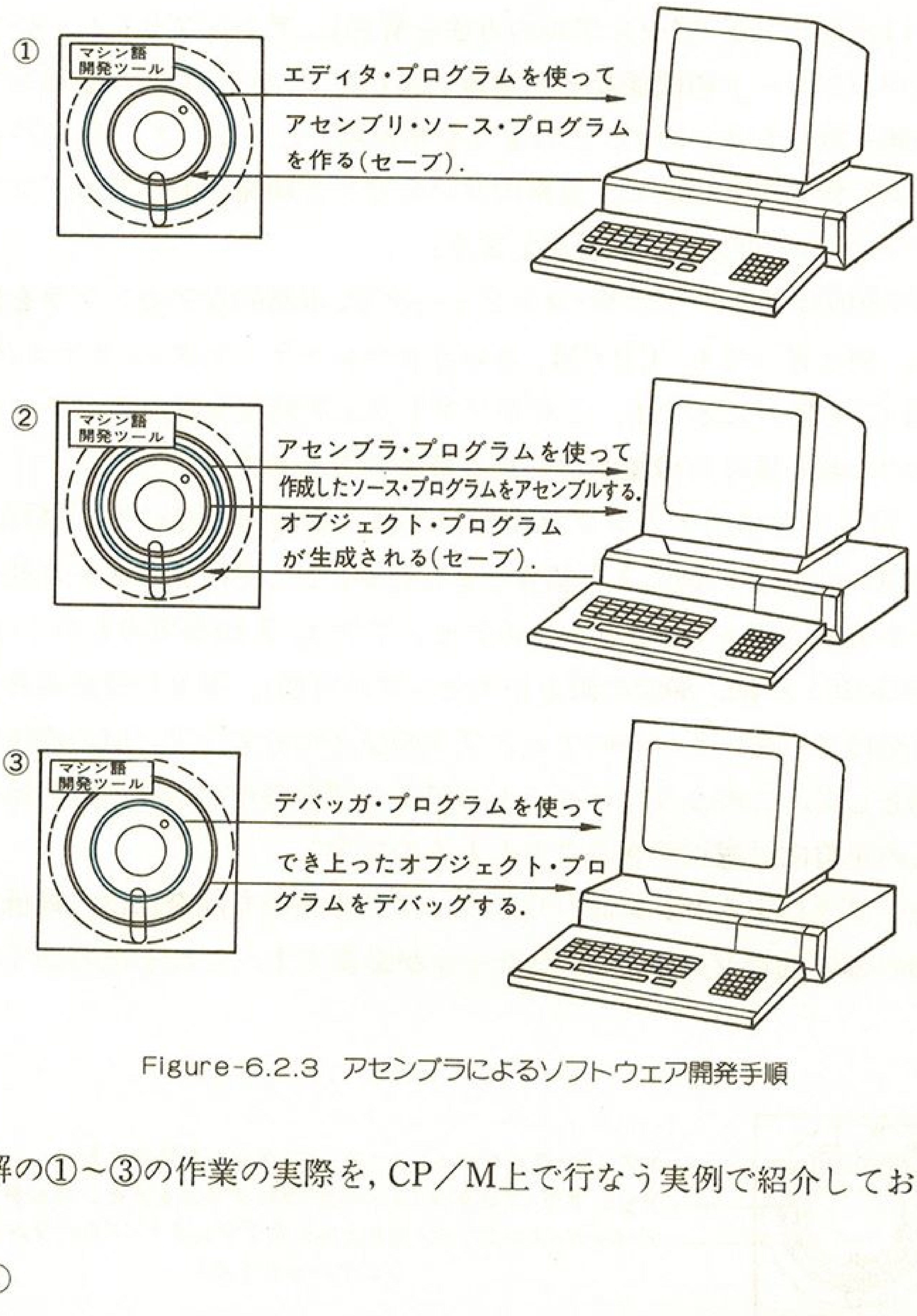


Figure-6.2.3 アセンブラによるソフトウェア開発手順

図解の①～③の作業の実際を，CP／M上で行なう実例で紹介しておきましょう．

手順①

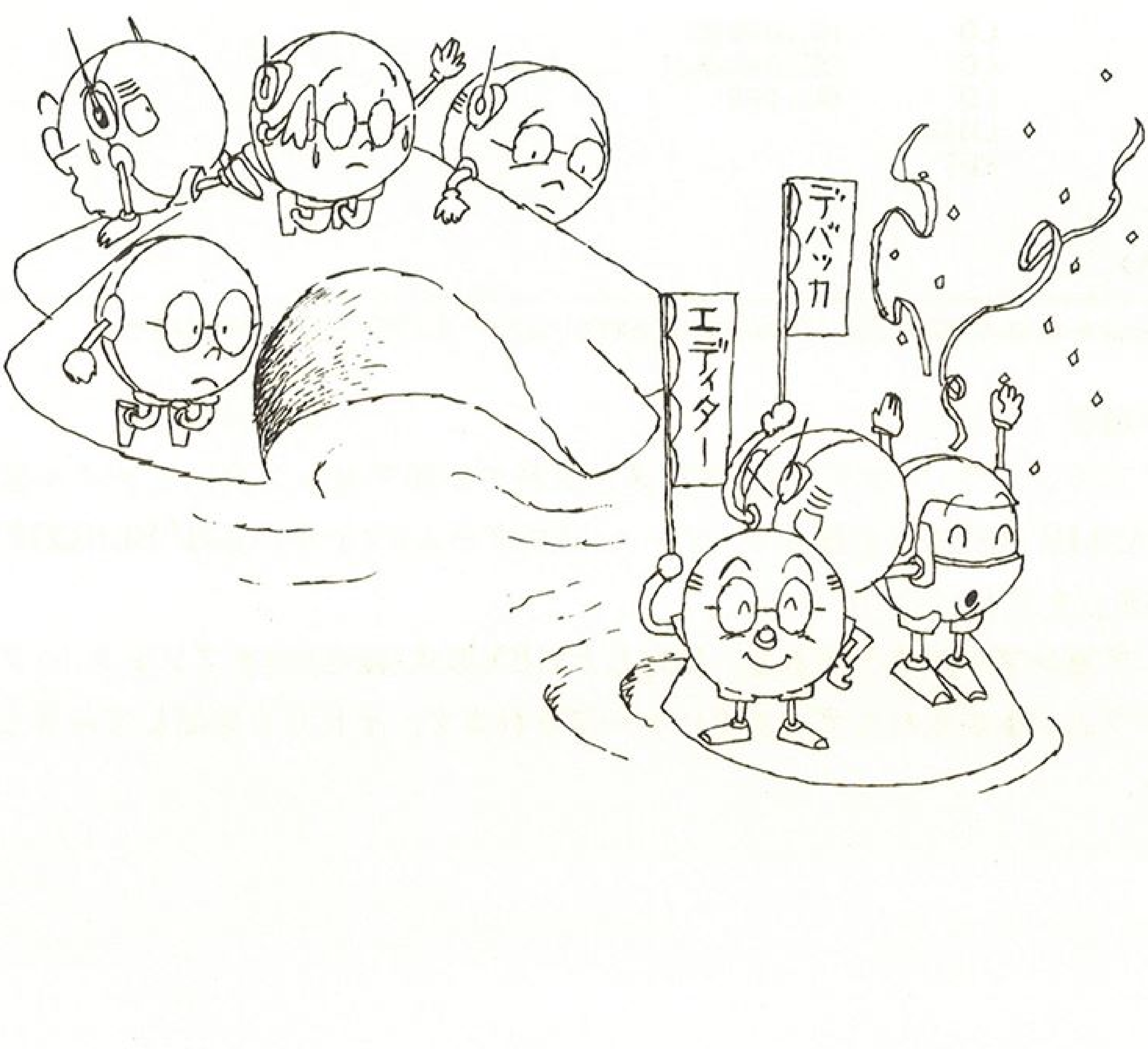
まずアセンブリ・ソース・プログラムを作らなければなりません．ソース・プログラムの作成には「エディタ」が必要です．この点は普通のパーソナル・コ

ンピュータに組み込まれているBASIC言語によるソフトウェア開発と大きく異なる点です．BASICが起動していればソース・プログラムの作成もでき，プログラムの実行もデバッグも同じBASIC上でできることはみなさんも経験していることでしょう．

しかしアセンブラは「ひとつのBASICインタープリタというプログラムで何んでもやってしまう」ものと基本的に異なっています．何んでも一緒にまとめてあるのは，パーソナル・コンピュータに組込みのBASICぐらいであると思ってよいでしょう．

アセンブラに限らずPASCAL, C, FORTRAN, COBOL, FORTHなどの「コンパイラ言語」と呼ばれる本格的なソフトウェアを作るための言語はすべて，ソース・プログラムをエディタで作成します．BASICでも，コンパイラ形のBASIC(マイクロソフト社の「BASCOM」や，デジタルリサーチ社の「CB-80」）はこれと同じです．

エディットに限らずアセンブルやコンパイル，それにデバッグなどもそれ



ぞれ別のプログラムによって行ないます.

さてアセンブリ・ソース・プログラムは適当なエディタ(たとえばCP/Mに付属の「ED」や，スクリーン・エディタの「ワード・マスター」など)を使って，すでに作成され，ディスク上にセーブされているとします．エディタの具体的な使い方については省略しますが，現在ディスク上にセーブされているファイル(ソース・プログラムとアセンブラのふたつ)のファイル名を確認し，念のためにソースファイルの内容をタイプアウトしておきましょう.

```
A>DIR
A: ASMB     COM : BLKCOPY  Z80

A>TYPE BLKCOPY.Z80
        ORG     9000H

        LD      HL,0000H
        LD      DE,0A000H
        LD      BC,100
        LDIR
        RET

A>
```

- A>DIR……ディスク上にセーブされているファイルのファイル名をタイプアウトせよ.
- ASMB COM：Z-80アセンブラ
- BLKCOPY Z80：アセンブリ・ソース・プログラム
- A>TYPE BLKCOPY.Z80………ファイル「BLKCOPY.Z80」(ソースファイル)の内容をタイプアウトせよ.
- ORG～RET：アセンブリ・ソースファイルの内容.（本節の例題と同じもの）

Figure-6.2.4　すでにディスクにセーブされているソース・プログラムとアセンブラ

手順②

アセンブラ・プログラム(クロメンコ社のZ-80アセンブラ，ファイル名は「ASMB．COM」)を使って，ソース・プログラム(ファイル名は「BLKCOPY．Z80」)をアセンブルします.

アセンブルが終了すると，指定したHEX形式(後述)のオブジェクト・プログラムが生成されてディスクにセーブされます．それらを確認してみましょう.

```
A>ASMB BLKCOPY HEX↵

CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

Errors                 0

end of assembly

A>
```

- ASMB ……アセンブルせよ.
- BLKCOPY ……ソース・プログラム「BLKCOPY」を.
- HEX ……生成するオブジェクト・プログラムはHEX形式で出力せよ.
- CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15 ……アセンブル作業開始のメッセージ.
- Errors 0 ……アセンブル・エラーなしのメッセージ.
- end of assembly ……作業終了のメッセージ.

Figure-6.2.5 アセンブラによるアセンブル

```
A>DIR↵

A: ASMB     COM : BLKCOPY  Z80 : BLKCOPY  HEX : BLKCOPY  PRN
```

- A>DIR ……ディスク上のファイルのファイル名をタイプアウトせよ.
- ASMB COM: アセンブラ
- BLKCOPY Z80: ソース・プログラム
- BLKCOPY HEX: 生成されたHEX形式のオブジェクト・プログラム.
- BLKCOPY PRN: 生成されたリストファイル.

```
A>TYPE BLKCOPY.HEX↵

:0C9000002100001100A0016400EDB0C9C7
:0000000000
```

- A>TYPE BLKCOPY.HEX ……生成されたオブジェクト・プログラムをタイプアウトせよ.
- :0C90… ……インテルHEX形式のオブジェクトファイル.

```
A>TYPE BLKCOPY.PRN↵
```

……生成されたリスト・ファイルをタイプアウトせよ.

CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

アドレス	オブジェクト・コード	ラインNo.	ソース・プログラム	
		0001	ORG	9000H
9000	210000	0002	LD	HL,0000H
9003	1100A0	0003	LD	DE,0A000H
9006	016400	0004	LD	BC,100
9009	EDB0	0005	LDIR	
900B	C9	0006	RET	

……アセンブルリスト

```
Errors                 0

A>
```

Figure-6.2.6 アセンブラにより生成されたオブジェクト・プログラムの確認

このようにアセンブラを実行したことにより，HEX形式のオブジェクトファイルと，リストファイルが生成されています．

HEX形式というのは，インテルHEX形式のことで，最も広く使われているオブジェクトファイルの形式です．先のHEXファイルをわかりやすく区分して説明したものを次に示します．

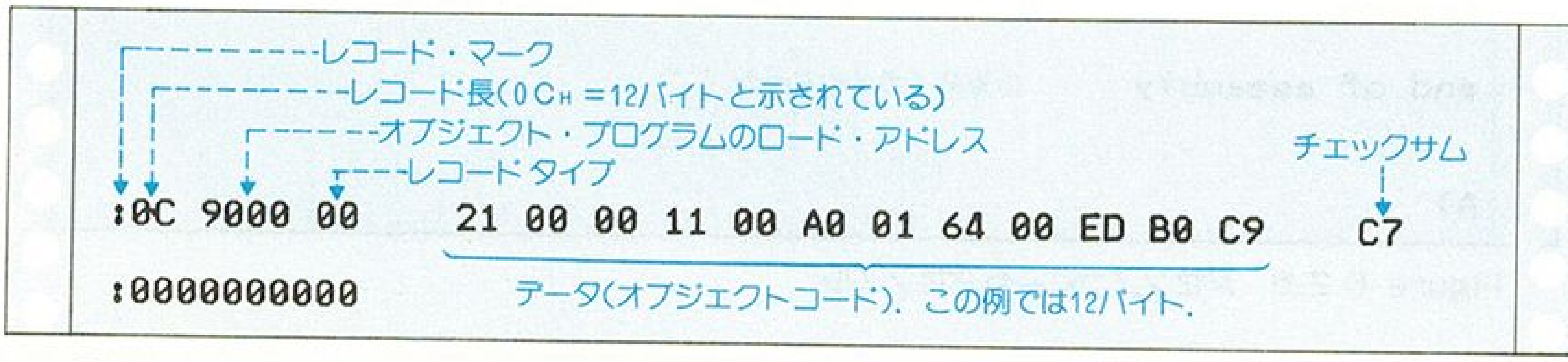


Figure-6.2.7 インテルHEX形式の説明

手順③

デバッグ作業に使用するツールとしては，デジタルリサーチ社の「DDT」（ダイナミック・デバッギング・ツール．これはCP／Mに付属している）や，その上位版の「ZSID」（Z-80シンボリック・インストラクション・デバッガ）が一般的ですが，その実行例の紹介は省略します．

以上，CP／M上の本格的アセンブラを使ったソフトウェア開発の作業実例を紹介しましたが，これらはごく基本的な作業の一例にすぎないことをお断りしておきます．

なお本章で例題としているアセンブリ・ソース・プログラムは，わずか5～6行（ステップと言う）のものですが，実際の実用ソフトウェアを作成する場合は，数十から数千ステップになります．ただし，普通は大規模なプログラムを作成する場合，アセンブリ・ソース・プログラムの段階でいくつかのブロックに分割して（モジュール化という）開発し，それぞれのモジュールごとにアセンブルし，できあがったオブジェクト・プログラムを最後に1本のプログラムにまとめるという方法を取ります．いろいろなことから，ひとつのアセンブリ・ソース・プログラムが，あまりに長いものは好ましくないのです．

7

コンピュータの中心CPU

いよいよマシン語の核心にはいっていきます．

コンピュータのいろいろな働きは，すべてCPUを中心に行なわれます．パーソナル・コンピュータなどに使用されているCPUは，「マイクロプロセッサ」と呼ばれており，マイクロコンピュータ(俗称マイコン)と言うのは，「マイクロプロセッサを使ったコンピュータ」の意味なのです．

本章ではこのCPUがどのようにメモリ上のプログラムを次々と実行していくのか，また，CPUを構成するレジスタ群とその機能など，CPUの働きの核心について解説していきます．

7.1

●プログラム実行のメカニズム

まずコンピュータの動作の基本であり，かつCPUで最も重要な働きである「プログラムの実行」のメカニズムについて解説しましょう．

メモリ上にプログラムを置き，それを順次読み出してCPUに取り込み，その命令を解析実行していく方式(ストアード・プログラム・シーケンシャル・コントロール方式)こそ，「フォン・ノイマン型」と呼ばれる現在のコンピュータの大原理なのです．

では，前章で取り上げた，12バイトでできているブロック転送プログラムを例に解説していきましょう．この12バイトのプログラムはすでにアドレス9000_Hからのメモリ上にストアされているものとします．その様子をもう一度ここに図示しますが，前章のFigure-6.1.3も参照してください．

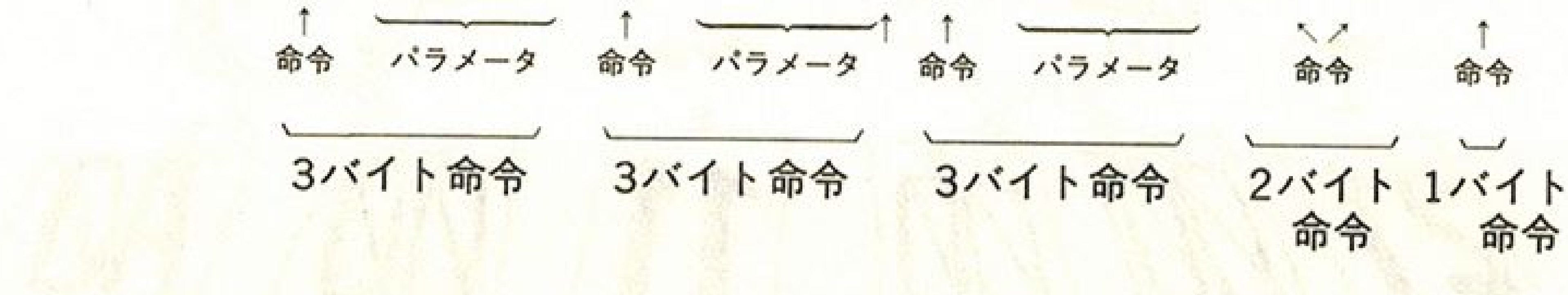


Figure-7.1.1 メモリ上のオブジェクト・プログラムの内容

メモリ上のプログラムは前図のように「命令」を表わすコードと，「パラメータ」を表わすデータが混在しています．図のメモリの色の付いている部分が命令を表わすコードであり，その他がパラメータです．

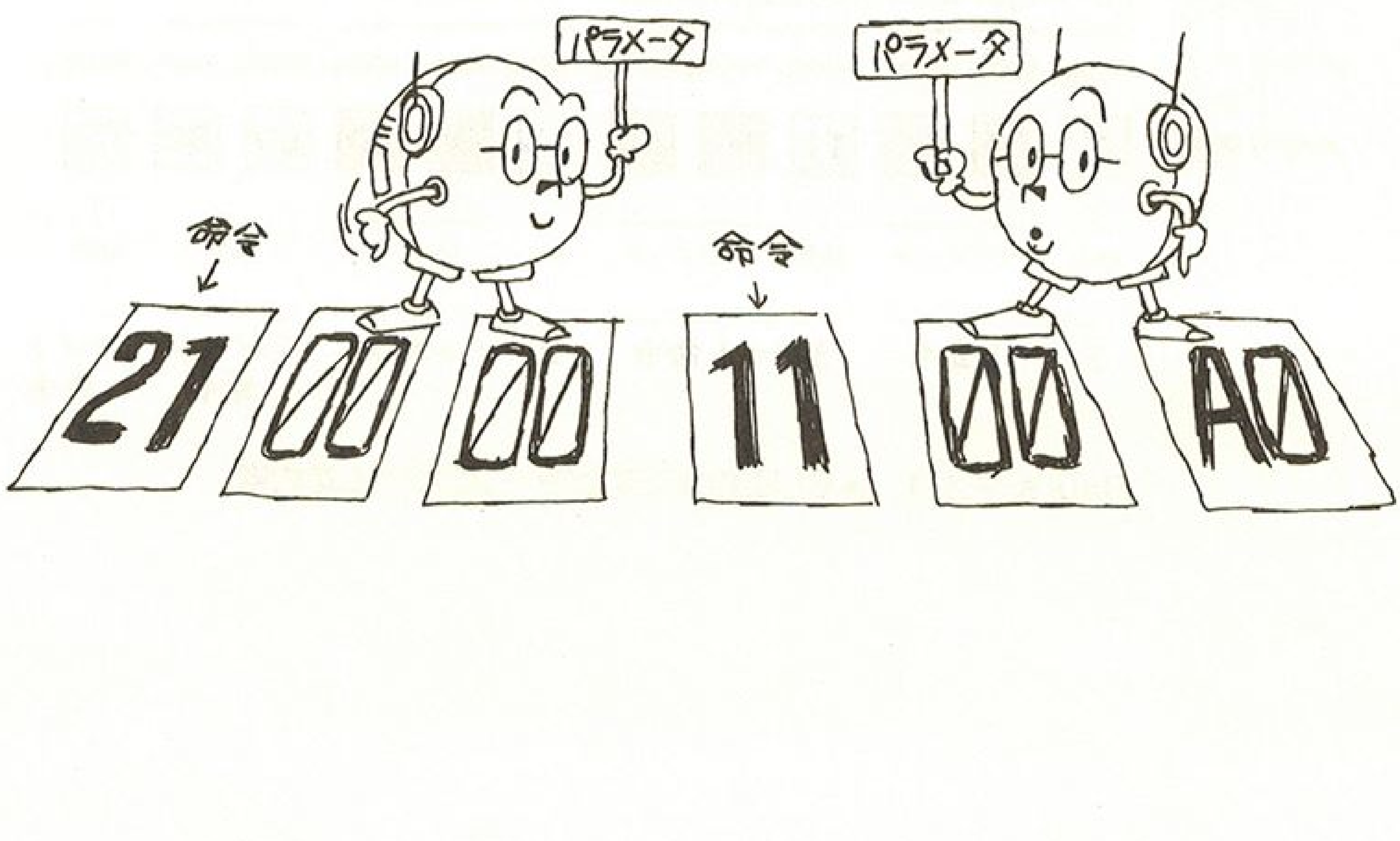
この図を基に，プログラム実行のプロセスを順を追って解説していきましょう．

まず，このプログラムは，アドレス9000_Hから実行されることを前提にメモリ上にストアされています．9000_Hから実行するには，本書の実習用に使用しているMCモニタのGコマンドを使えばできますし，各機種に独自の「モニタ」が組み込まれていれば，その中のマシン語プログラム実行コマンドを使ってもできるでしょう．

MCモニタのGコマンドで，G9000↵ のように実行すれば，アドレス9000_Hからプログラムの実行が始まります．なぜ始まるかはここでは問わず，とにかく9000_Hからの実行が開始されたとします．

まずCPUは，アドレス9000_Hのメモリ内容を読み出して，CPU内部に取り込みます．CPUは取り込んだ１バイトのマシンコードの命令を解析し，その結果に従った動作を開始します．

この一連の動作について，アドレスとメモリ内容とCPUの働きの関係を次の表に示し解説しましょう．この表から「プログラムの実行」とはどういうことなのか，そのプロセスの概要がつかめるのではないかと思います．



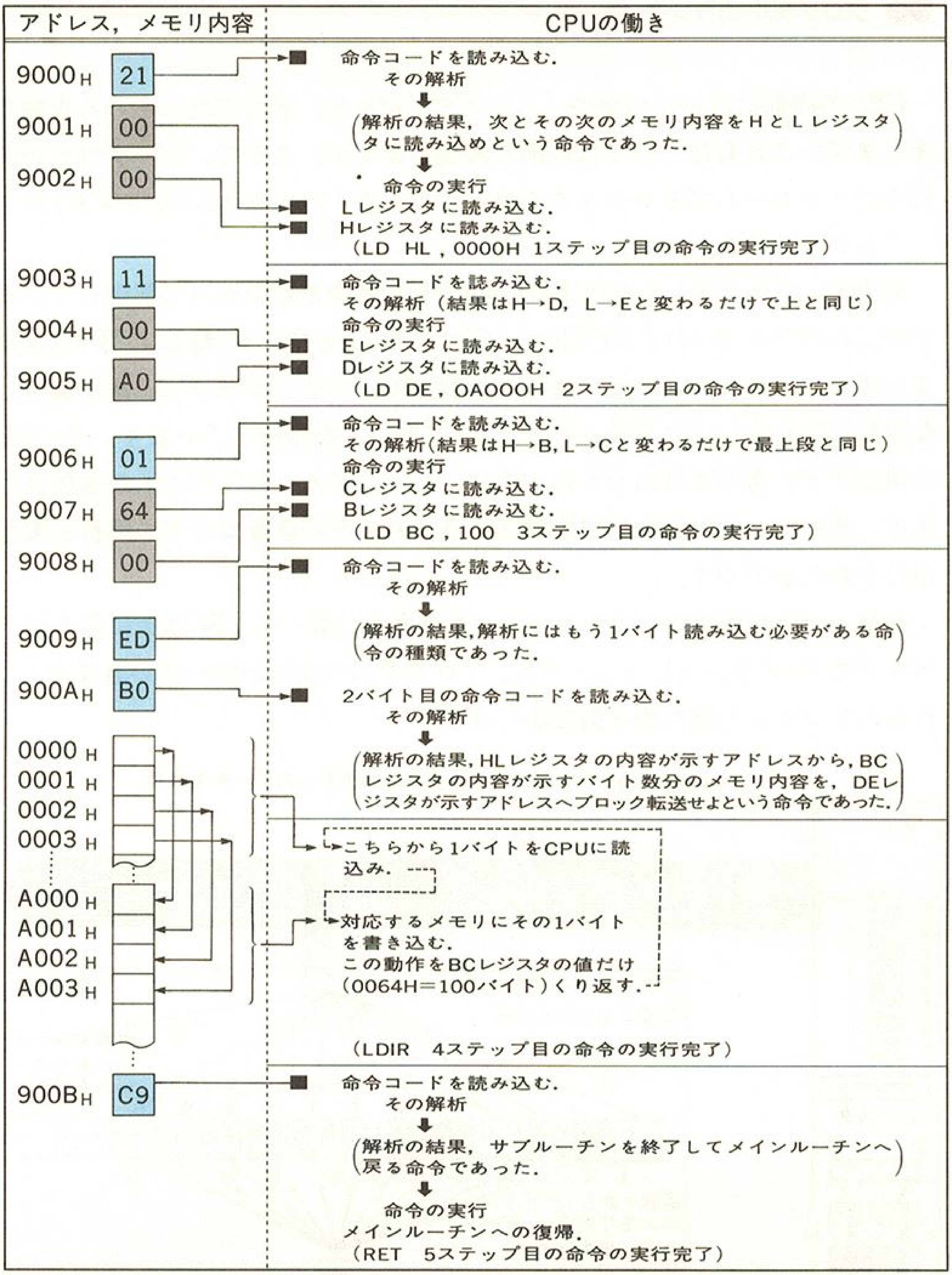


Figure-7.1.2 プログラム実行のプロセス

●● プログラム・カウンタ ●

CPUの内部にはいくつかの「レジスタ」があり，それぞれいろいろな働きをします．これらについては次節で解説しますが，ここで，プログラムの実行をコントロールするレジスタのひとつである「プログラム・カウンタ」についてふれておきましょう．

前表で，プログラムを実行する際のCPUの動作を順を追って解説していますが，このプロセスは，CPU内の「プログラム・カウンタ(略して"PC")」によってコントロールされています．PCは16ビットのレジスタで，次に実行しなければならないプログラムのメモリ・アドレスを保持しています．このPCの値がアドレスバスに出力され，それによってメモリの1バイトが選択されます．そして，その内容がCPUに読み込まれ，その命令コードの解析と実行が行なわれるのです．

命令コードの解析が行なわれると，その命令に従って，PCは次に読み出すべきメモリ・アドレスにセットされ，プログラムの実行が続いていきます．これらのプロセスを次の表と図で示します．

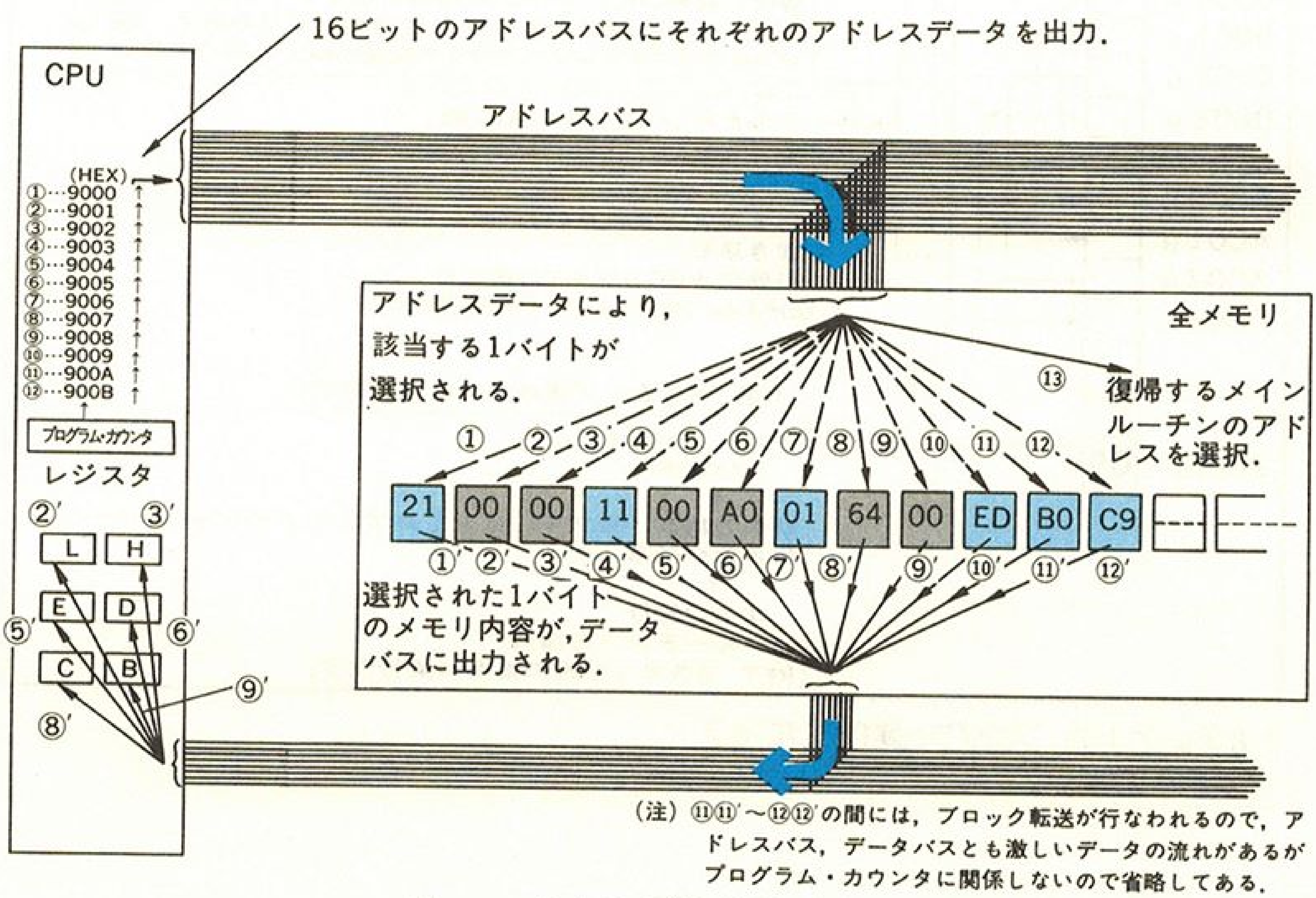


Figure-7.1.3 プログラム・カウンタの動き(1)

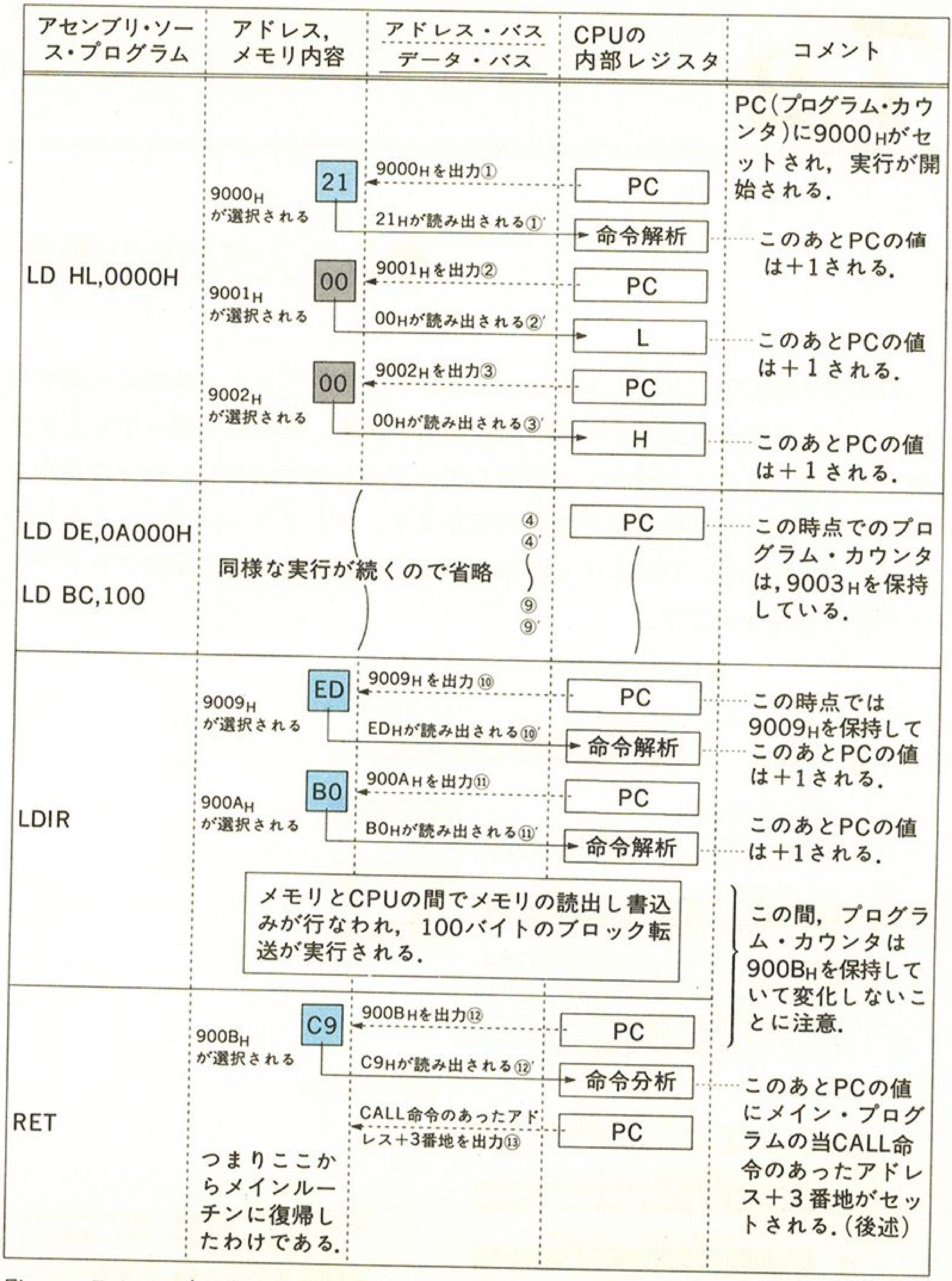


Figure-7.1.3 プログラム・カウンタの動き(2)

7.2

●レジスタとその働き

CPUの内部には，8ビットあるいは16ビットのデータを一時的に記憶する「レジスタ」と呼ばれるメモリ部が何組かあります．それらを使ってレジスタ対メモリ間，レジスタ対レジスタ間でデータの転送や演算など種々の操作をプログラム上で自由に行なうことができます．プログラムによってこれらのレジスタを操作し，目的とする結果を得ることこそ，マシン語のプログラミングそのものなのです．

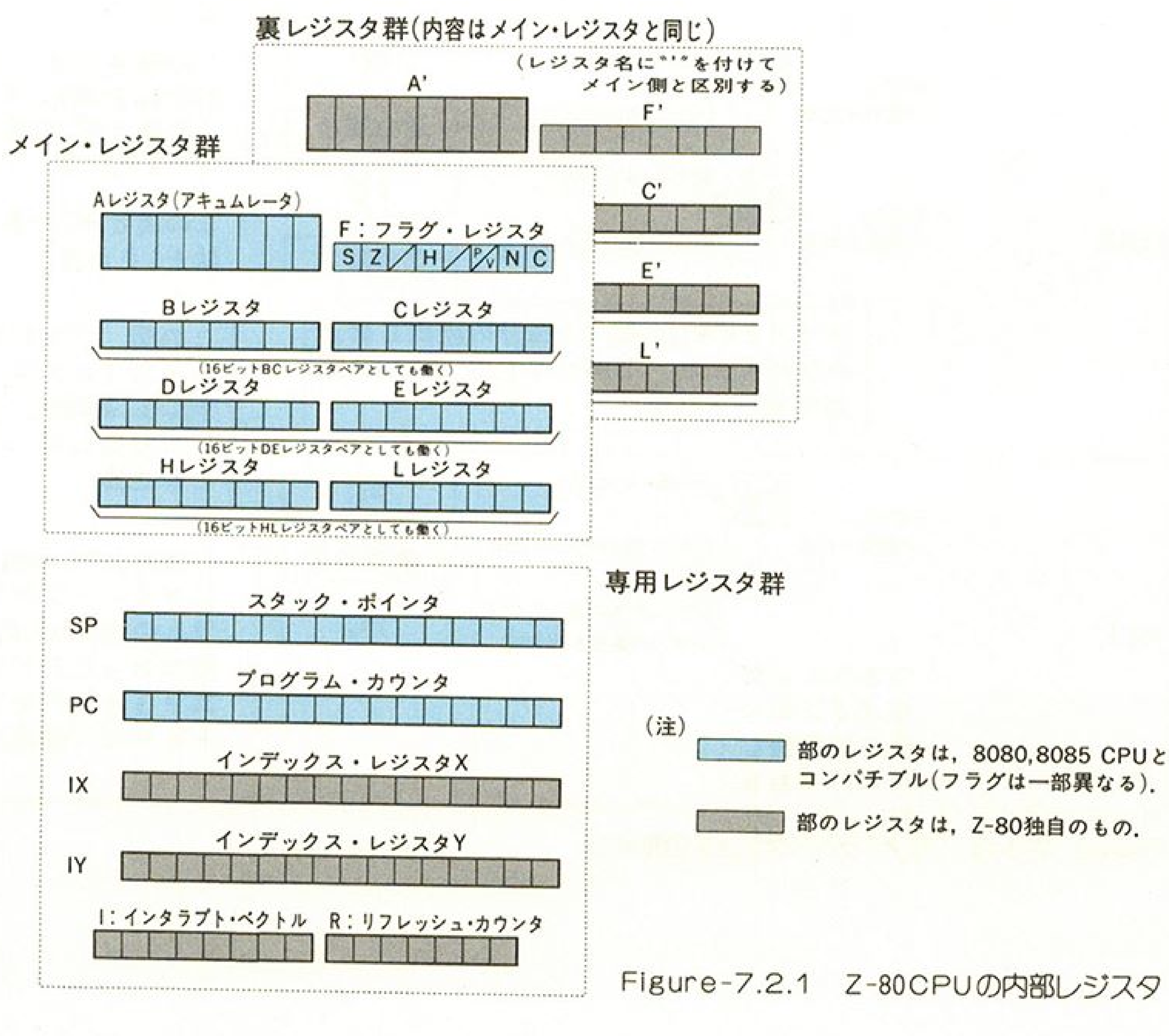


Figure-7.2.1　Z-80CPUの内部レジスタ

まず，Z-80CPUのレジスタの構成図を示しました．Z-80と8080および8085の関係は，図の色分けで示してありますが，両者に共通のレジスタのみを使用すれば，Z-80,8080,8085いずれのCPUでも実行可能なプログラムを作ることができます．

プログラムによって私たちが使用可能なレジスタは，図に示されている計22個であり，これらはメイン・レジスタ群，裏レジスタ群，専用レジスタ群の3つに分類されます．

●● メインレジスタ群 ●

A,B,C,D,E,H,Lの各レジスタは，どのような目的にどう使おうと自由な「汎用レジスタ」です．それぞれは8ビットのレジスタですが，図のように「BC」，「DE」，「HL」はそれぞれを1本にまとめ，16ビットのレジスタとして扱うこともできます．その場合は，「BCレジスタペア」，「HLレジスタペア」などと呼びます．

図ではAレジスタだけが少し大きく描かれていますが，これはAレジスタだけが他のレジスタにはない特別の機能を合わせ持っていることを表現しています．Aレジスタは，加·減算，論理和·積，排他的論理和，比較などの演算ができる，全レジスタの中で唯一のレジスタであり，最もよく使用される重要なレジスタで，「アキュムレータ」とも呼ばれています．

このメイン·レジスタ群は，フラグの一部が若干異なるだけで，8080および8085CPUとはコンパチブルです．

●● フラグ·レジスタ ●

メイン·レジスタ群の中で，このフラグ·レジスタだけは性質が異なり，各種の「フラグ」の状態を示すために専用に使われているレジスタで，他のA,B,……Lレジスタのようにプログラム上から自由に操作することはできませ

ん．フラグとは合図をする「旗」の意味で，各種の演算や比較の結果が正であるか負であるか，０であるか否かなどの判別結果を知らせるためのものです．それぞれの項目に対する判別結果は１ビットの０か１で示されます．この判別結果は，演算や比較などを行なったときに自動的に書き換えられ，このフラグを利用したプログラムにより各種の条件分岐などが行なわれるわけです．詳しくは次の章で実習解説します．

●● 裏レジスタ群 ●

裏レジスタの構成は，メイン・レジスタの構成とまったく同じです．しかし裏レジスタをプログラム上から直接使用することはできず，使用する際は「レジスタ交換命令」で，裏レジスタの内容とメイン・レジスタの内容を入れ替えてから，メイン・レジスタ上で使用します．用が済めばもう一度「レジスタ交換命令」を実行して最初の状態に戻すことができます．これはミニ・コンピュータでよく使われているアーキテクチャをマイクロプロセッサに取り入れたもので，8080，8085CPUにはないレジスタ群であり，いろいろな応用が考えられます．

●● 専用レジスタ群 ●

専用レジスタは，その使用目的や機能がもとから定まっているレジスタであり，プログラマーの意志で何にでも使えるメインや裏レジスタなどの汎用レジスタと異なっています．

専用レジスタで最もよく使われるのは「スタック・ポインタ(SP)」であり，前節でふれている「プログラム・カウンタ(PC)」と共に，次の章で実習解説します．

これらのレジスタをどう働かせるかについては，次の章で実際にプログラムを組んで実行しながら解説していきます．

8

マシン語命令の実習

8

本章からマシン語そのものの実習にはいります．みなさんがMCモニタを使ってメモリに書き込んだプログラムを実行し，その結果を確認しながら各命令の働きを解説していきます．マシン語の一つひとつの命令はどれも単純で明解です．

この単純明解な命令を組み合わせていくことにより，非常に高度なソフトウェアをプログラミングすることも可能になるのです．一つひとつの命令をいかにうまく組み合わせて目的を達成できるプログラムを組むかは「プログラミング技法（テクニック）」の問題であり，これは本書を卒業した次の段階では重要なこととなります．

本章では，マシン語の基礎である個々の命令について解説しますが，マシン語の応用について多くを取り上げているわけではありません．まず個々の命令を学習することにより，CPUの「しくみと働き（アーキテクチャ）」を理解することが最も大切なのです．あるひとつの命令「LD A,B」がマシン語のコードで「78」であるというようなことを覚えるのが目的ではありません．

ここではスクリーンにインベーダーこそ飛び回りませんが，目先のことにとらわれずに，まずCPUとマシン語の考え方の基礎を学んでください．

8.1

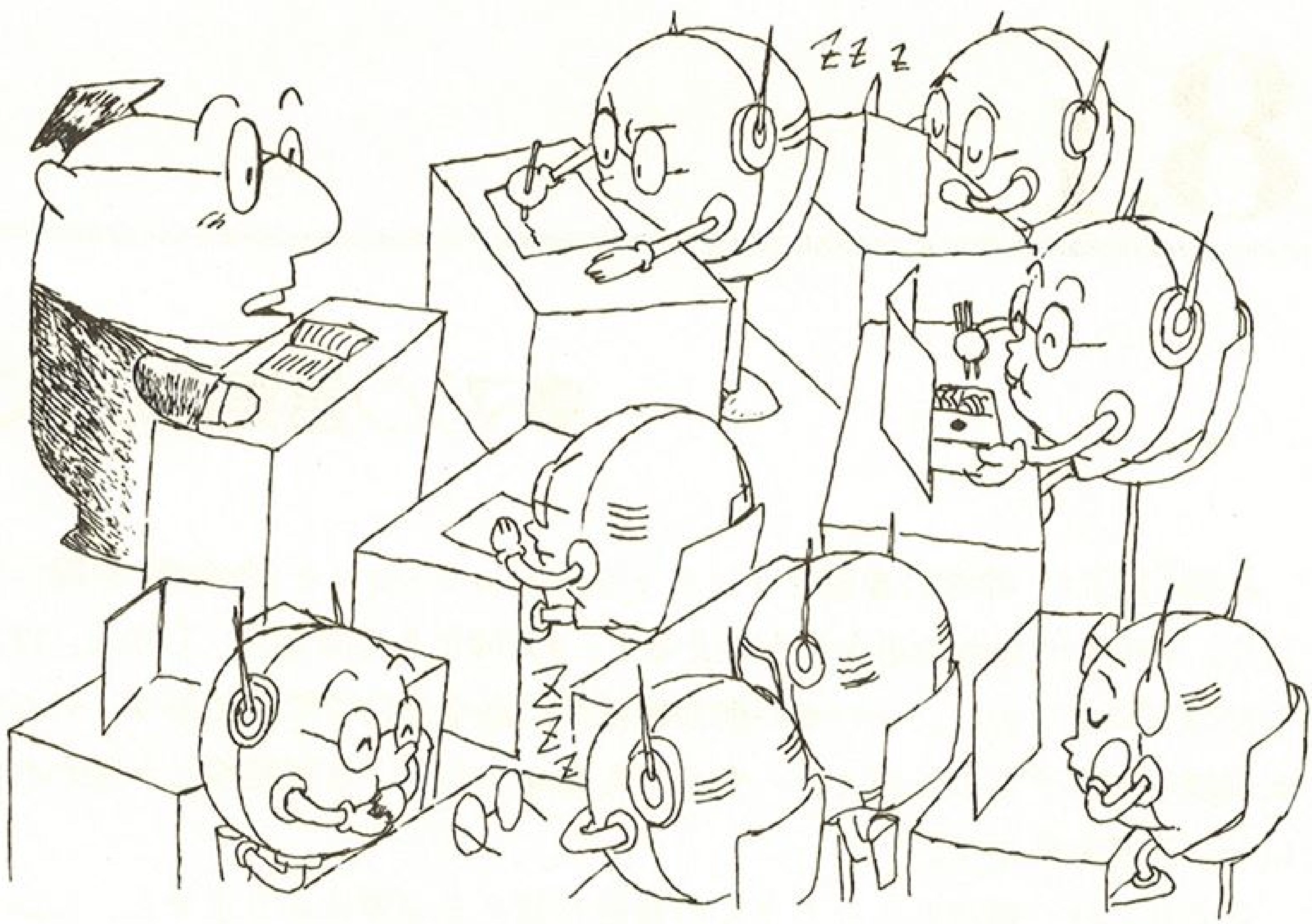
●マシン語実習の前に

Z-80CPUは，非常に豊富なインストラクション・セット(命令群)を持っており，すべての命令の組合せを数えると，約700にもなります．しかし，ほとんどのプログラムは，せいぜい何10種類かの命令だけでできており，いかに大規模なプログラムといえど，多くても全命令の１～２割程度しか使われていないと思われます．

そんなわけで最初からたくさんの命令を覚える必要はありません．たぶん20～30も使えれば，多くのプログラムは組むことが可能となるでしょう．さらに次のような事実も知っておいてください．

CP／Mベースのビジネス用アプリケーション・ソフトのベストセラーを例にとると，簡易言語の「SuperCalc(スーパーカルク)」や「Multiplan(マルチプラン)」，リレーショナル・データベースの「dBASEII(ディーベース・ツー)」，「Personal Pearl(パーソナルパール)」などの〝実務に耐える〟優秀なソフトウェアなども，Z-80の1/3しか命令を持たない8080CPUの命令のみで作られているのです．

これらのことからも，まずは，重要ないくつかの基本的な命令を使えるようになることが先決です．基本的な命令だけの組合せでも，プログラムは十分に組めるのです．華麗なテクニックを使って，プログラムが占めるバイト数を少なくしたプログラム(オブジェクト効率のよいプログラム)が優れたプログラムというわけではありません．また，バイト数が少ないから実行速度が速いプログラムであるとも言えません．マシン語の学習を始める最初から心掛けなければならない大切なことは，高度な命令を駆使することではあり



ません．基本的命令でよいから，「きちんと整理された，わかりやすいプログラムを組むこと」が第一なのです．

本書では，プログラミングの基本的命令となっているいくつかの重要な命令について解説します．これらをしっかり身に付ければ，その他の命令はその場合場合によって使うメリットがあるものからしだいに取り入れていけばよいでしょう．

インストラクションのすべての組合せはZ-80で約700，8080，8085で約250ぐらいありますが，これらを機能別のグループに分け，わかりやすく整理した一覧表が，各CPUの「インストラクション・セット一覧表」と言われるもので，各CPUの製造会社からも発売されています．みなさんがアセンブラを使い始めると，座右の表として参照することになるものです．

しかし，だんだん経験を積んでいき，セミプロ・クラスの力が備わってくると，よほど特殊な目的の仕事でない限り，この一覧表を見ない人が多くなる

ようです．これは，通常のプログラムを組む場合，人によって多少の差はありますが，使う命令の種類がだいたい固定されてくるからなのです．自分にとって使い慣れたなじみの命令や，何となく気に入っている命令を，どうしても多く使うようになるものです．またほとんどのプログラムは，これらのよく知っている命令だけで十分作成することができるのです．

本章で大切なことは，各CPU独自の特別な機能を追求するより，コンピュータの基本動作に関わる一般的な基本命令を正しく理解することです．スクリーン上のインベーダーを少しでも高速に動かしたいとしても，そこの部分だけのマシン語を知りたいなどというのではなく，その前にコンピュータの基本的なアーキテクチャを理解してほしいのです．そうすればインベーダーはあなた自身の力で自由に飛び回ることになるでしょう．

巻末にZ-80CPUの多くの命令群の中から，本書で取り扱う基本的な命令を，機能グループ別にまとめ，「機能別主要命令表」として示してあります．本章では，それをグループ別に取り上げ，MCモニタによる実習を中心に解説を行なっていきます．

「Z-80の基本的な命令」とは，8080CPUと共通の命令を主体に，Z-80独自の命令の中からも一般的によく使われるいくつかを取り上げたもので，筆者の選択によるものです．Z-80と8080CPUのレジスタの共通部分については前章のFigure-7.2.1でふれています．

今からマシン語プログラムの実習を行なっていきます．本書に示されている実行例等は，PC-8001／8001MKIIおよびPC-8801のN-BASIC上のフリーエリアであるアドレス9000_H台を使用し，ここに実習プログラムを書き込んで実行しています．他の機種ではこのフリーエリアが9000_H台から使用できないものもあるため，実習プログラムの一部を変更して，$D000_H$を使用したりします．詳しくは巻末APPENDIX 1を参照してください．

8.2

●8ビット・ロード命令

「8ビット・ロード命令」とは，Figure-7.2.1のZ-80CPUの内部レジスタの図で，メイン・レジスタ群中のA, B, C, D, E, H, Lの8ビットの各レジスタに関する命令で，各レジスタ間で行なう1バイト単位(8ビット単位)のデータのやり取り，およびメモリと各レジスタ間で行なう1バイト単位のデータのやり取りを行なう命令群です．

メモリとCPUのレジスタ間で行なうデータのやり取りに関しては，言葉の上からは習慣として，

レジスタ ⇦ メモリの場合を「ロード」
レジスタ ⇨ メモリの場合を「ストア」

と呼んでいます．しかしマシン語の命令では，どちらの場合でもすべて「LD(ロード)」だけですので注意してください．

ここで本書で使っている「ロード」，「ストア」，「セーブ」，「転送」の意味をまとめておきましょう．

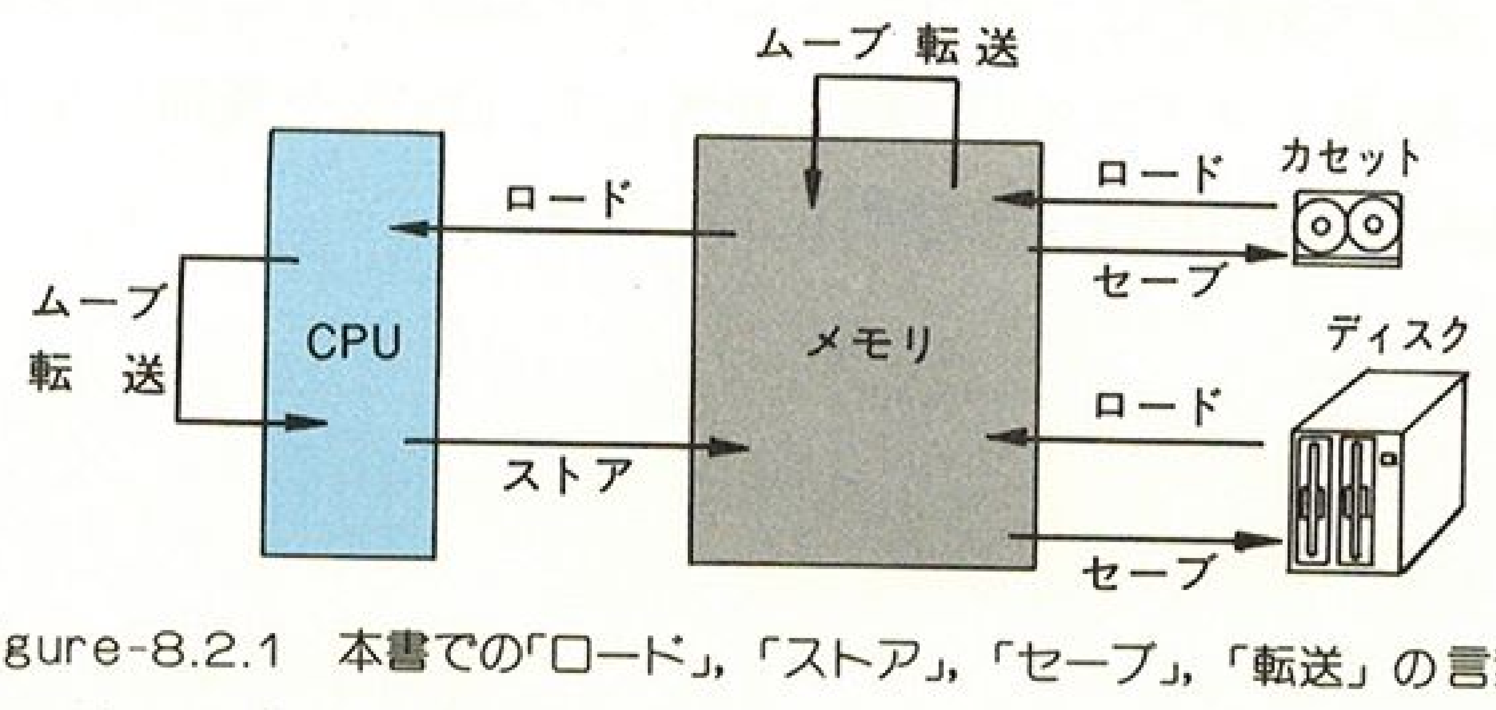
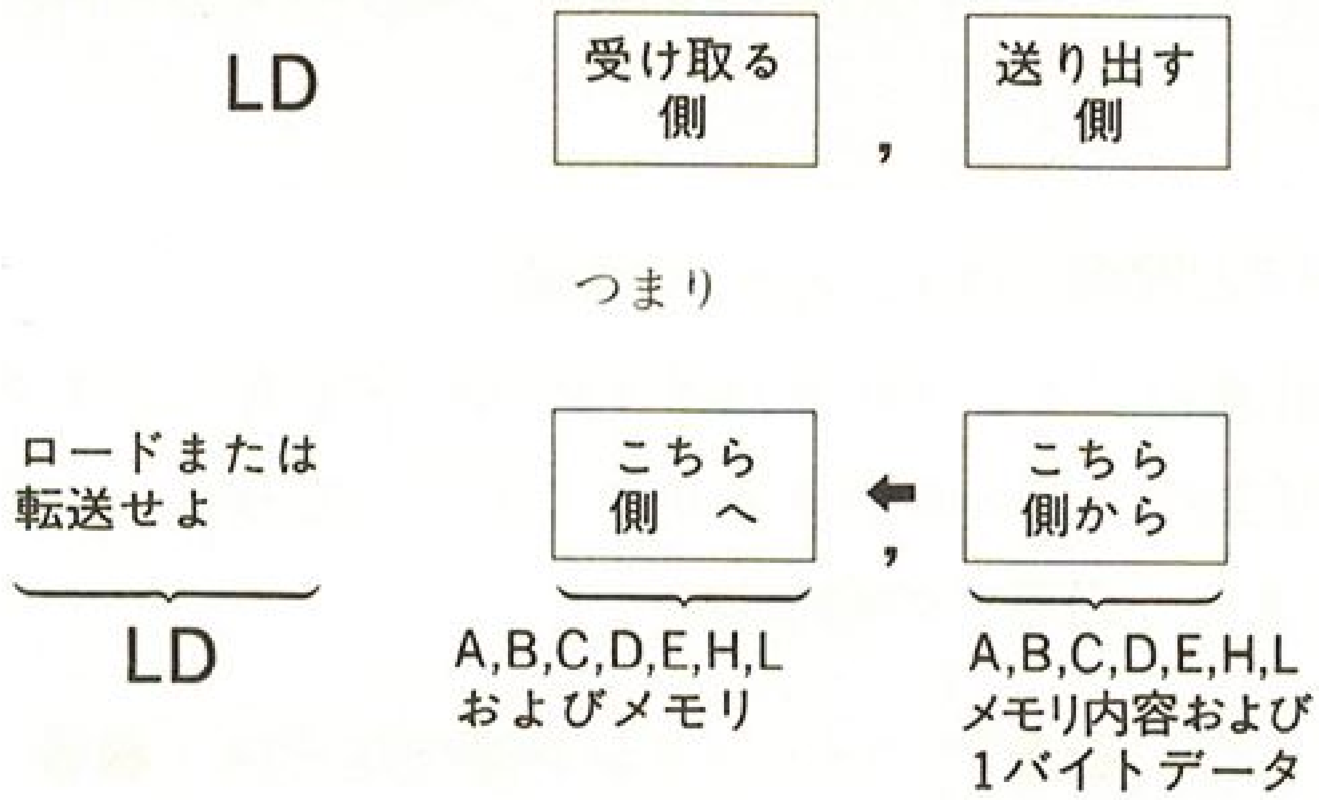


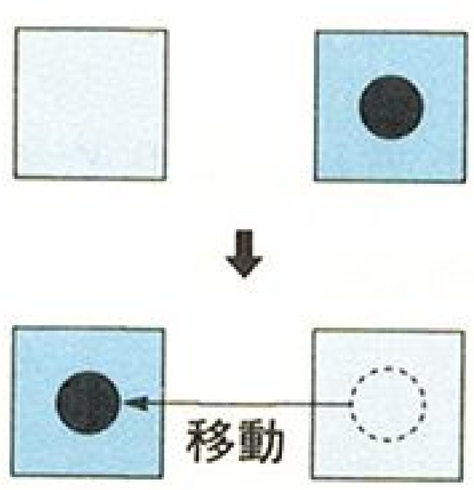
Figure-8.2.1　本書での「ロード」，「ストア」，「セーブ」，「転送」の言葉の使い方

ロード命令は転送命令とも呼ばれますが，次のような形式で表わします．

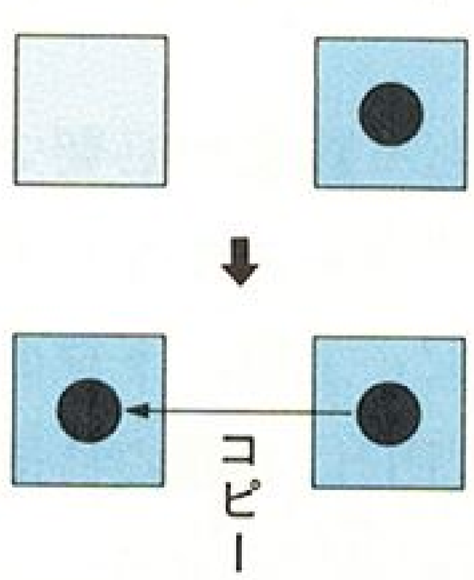


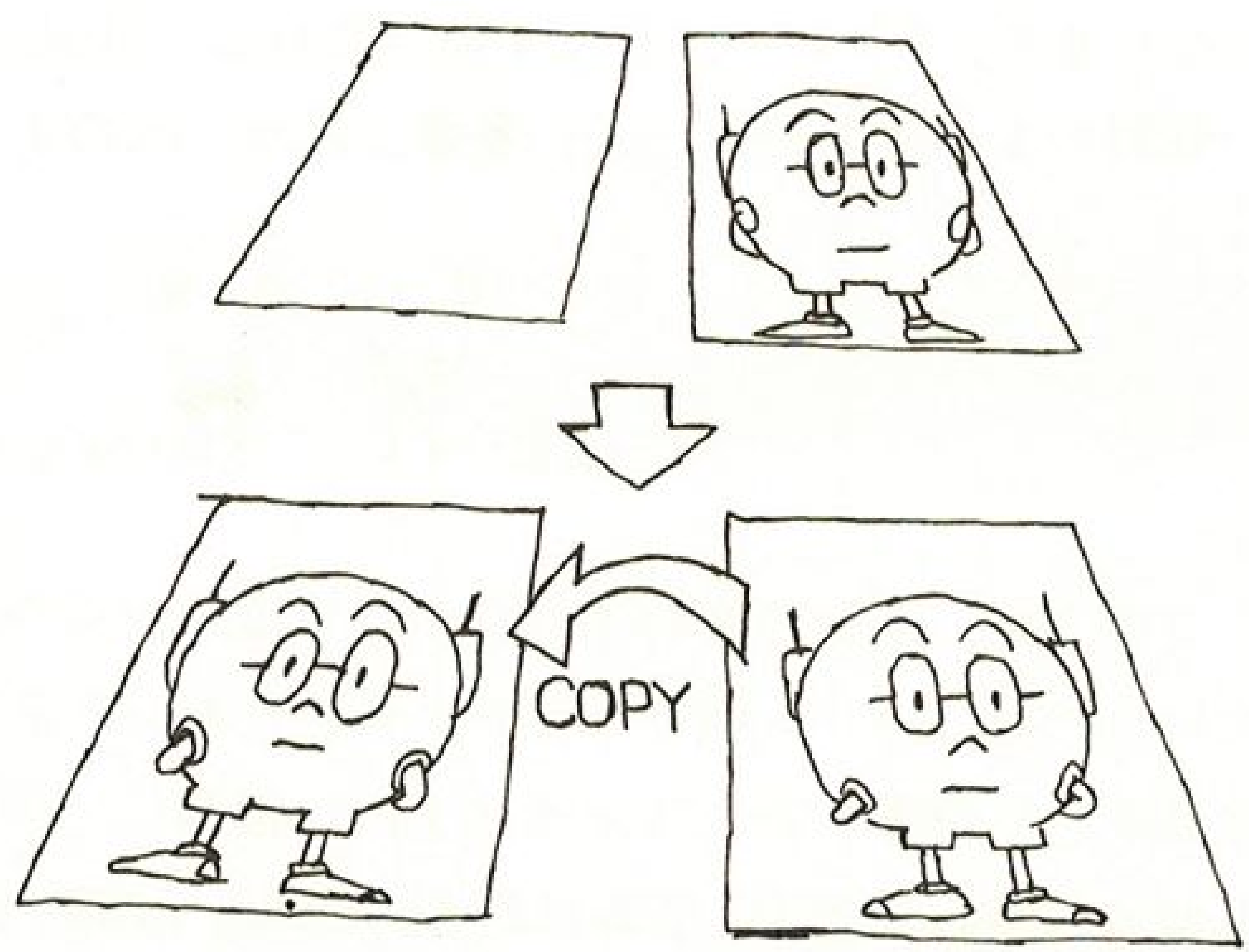
このようにロード命令は，のちほど解説する16ビット・ロード命令も含めて，データをロード，ストアあるいは転送する命令であり，そのデータの流れの方向は命令式の上ではすべて「⇦」向きです．

ここで誤解しないよう注意しておくことは，「ロード」や「転送」の意味は，



ということではなく，





という意味であり，ロードしたあと右側のもとのデータがなくなってしまうわけではないということです．

では，8ビット・ロード命令の代表的なもののいくつかを実習によって解説していきましょう．当命令の主要なものの一覧表は巻末にありますので参照してください．

実習1　メモリ間の1バイトデータの転送

メモリ上の任意のアドレスの1バイトのデータをAレジスタにロードし，そのまま別の任意のアドレスのメモリにストアしてみましょう．

「Aレジスタ⇦メモリ内容」の命令は，

LD　A，（○○●●）*　………マシン語では「3A　●●　○○」の3バイト．

（○○●●：2バイトのアドレス値）

（巻末の機能別主要命令表を参照）

という形式で表わします．たとえばアドレス104_Hの内容をAレジスタにロードするのは，

LD　A，(0104H)………マシン語では　3A　04　01

となります．アドレス104_Hは4桁で表わして0104_Hとした方がわかりやすく，一般形の2バイトの値「○○●●」とは，次のように対応します．

0 1	0 4
○○	●●
上位バイト	下位バイト

この上位，下位の各バイトについては，少々ややこしく注意が必要です．というのは，マシン語としてメモリ上に並ぶときは，アドレスや2バイトの数値が○○●●であればメモリ上には●●○○のように，上位，下位のバイトが入れ替わります．これはZ-80や8080，8085，6502CPUに共通の特徴になっています．

さて，上の3バイトの命令を実行すると，Aレジスタにはアドレス104_Hのメ

*マシン語にする時に○○●●→●●○○となることに注意．

モリの内容がロードされることがわかりました．

次はそのAレジスタのデータをそのまま任意のメモリにストアする命令です．

「メモリ⇦Aレジスタの内容」の命令は，

LD　(○○●●)，A………マシン語では「32　●●　○○」の3バイト．
（○○●●は2バイトのアドレス値）
（巻末の機能別主要命令表を参照）

という形式で表わします．たとえばAレジスタの内容をアドレス9100_Hのメモリにストアするのは，

LD　(9100H)，A………マシン語では　32　00　91

となります．ここでも「9100」はメモリ上にロードしたとき，「00　91」と上位，下位のバイトが逆になりますので注意してください．このことはLD命令に限りません．すべての2バイトデータはメモリ上では逆になると思ってください．

よって「実習1」は次のようなソース・プログラムになります．

```
LD   A,(0104H) …アドレス104Hのメモリ内容をAレジスタにロードする．
LD   (9100H),A …Aレジスタの内容をアドレス9100Hのメモリにストアする．
RET  ………………当サブルーチンを終了してMCモニタに戻る．
```

最後のRET命令は，コール，リターン命令の項で解説しますが，MCモニタからGコマンドで実行すると，サブルーチンコールとして実行されるため，実行終了後にもとのMCモニタに戻るために必要な命令なのです．

上の3ステップのプログラムを，ロード・アドレス（プログラムとしてメモリ上にロードする場合のスタート・アドレス）を9000_Hとしてアセンブルしてみましょう．これならアセンブラを使うよりハンド・アセンブルの方が早いですね．

ロード・アドレス	オブジェクト・プログラム		ソース・プログラム
			ORG 9000H*
9000_H	3A 04 01	←	LD A,(0104H)
9003_H	32 00 91	←	LD (9100H),A
9006_H	C9	←	RET

このように7バイトのオブジェクト・プログラム(マシン語)ができました。これを実行するにはアドレス9000_Hからのメモリ上に,

3A	04	01	32	00	91	C9
↓	↓	↓				↓
9000_H	9001_H	9002_H	……	……	……	9006_H

という順に書き込みます。

```
-D100 104↲ ………アドレス100H ～104H のダンプ.
0100   00 CD 7C 01 3E
                   └──アドレス104H のデータを確認しておく(値は機種により異なる).
-S9000↲*
9000 47 3A↲
9001 F0 04↲
9002 A1 01↲
9003 D3 32↲   Sコマンドでアドレス9000H からオブジェクト・プログラムを書き込む.
9004 10 00↲
9005 F3 91↲
9006 3A C9↲
9007 01 .↲    ………「.」でSコマンドを終わる.
(ここの値は不定.)
-D9000 9006↲………ダンプしてそれを確認.
9000  3A 04 01 32 00 91 C9
      書き込んだオブジェクト・プログラム.
-S9100↲
9100 E6 00↲
9101 01 00↲   結果がストアされるメモリを事前に0クリアしておく(特に必要ではないが).
9102 28 00↲
9103 FA .↲    ………Sコマンドを終わる.
-D9100 9102↲ ………ダンプして0クリアを確認.
9100  00 00 00

-G9000↲ ………実習1のプログラムを実行する. スタート・アドレスは9000H.

-D9100 9102↲………実行結果をダンプして確認.
9100  3E 00 00
      ↑
-     アドレス9100H にアドレス104H のメモリ内容(3EH)がストアされている.
```

Figure-8.2.2 実習1(メモリ間の1バイトデータの転送)の実行

Figure-8.2.2のようにMCモニタを使って，この7バイトのオブジェクト・プログラムをメモリに書き込み，その後スタート・アドレスである9000_Hから実行して結果の確認を各自で試みてください．

実習2 1バイトのデータをメモリ上に書き込む

任意の1バイトデータ(00_H〜FF_H)をAレジスタにロードし，そのまま任意のメモリにストアしてみましょう．

「Aレジスタ⇦1バイトデータ」の命令は，

LD A,●● ………マシン語では「3E ●●」の2バイト．
（●● ＝ 1バイトデータ）

という形式で表わします．たとえば1バイトのデータ「FF_H」をAレジスタにロードするのは，

LD A,0FFH………マシン語では 3E FF

となります．今回の命令は前回のものと違って( )が付いていませんね．これは間接的な表現ではなく，「FF_H」という数値そのものが直接Aレジスタにロードされることを表わしています(0FFHの0については6.2章の「ハンドアセンブル」を参照)．

ここでオペランドの( )について解説しておきましょう．これらはのちほどの実習でも出てきますが，とりあえず一例を挙げて説明しておきます．

LD A,(0104H) …( )が付くと間接的表現となり，値「0104_H」そのものではなく，値「0104_H」によってアドレスされるメモリの内容をAレジスタにロードする．

LD A,0FFH ……( )がないと直接的な表現となり，値「FF_H」そのものをAレジスタにロードする．

＊9000_H番台のエリアをフリーエリアとして使用できない機種では，マークされている9をDに変更する必要がある．以後のリストも同様．

というようになります．(　)が付くと(　)の中身そのものではなく，中身によってアドレスされるメモリの内容が対象，(　)がないと，その数値そのものが対象となるのです．まったく意味が異なりますので十分に注意してください．

さて実習プログラムに戻りましょう．

「メモリ⇦Aレジスタ」の内容の命令は「実習１」と同じですが，今回はアドレス9101_Hにストアしましょう．よって「実習２」は次のようなソース・プログラムになります．

```
LD   A，0FFH ………Aレジスタに１バイトデータ「FFH」をロードする．
LD   (9101H)，A……Aレジスタの内容をアドレス9101Hのメモリにストアする．
RET  …………………当サブルーチンを終了してMCモニタに戻る．
```

これもハンド・アセンブルします．スタート・アドレスは今回も9000_Hとしましょう．結果は次のようになります．

ロード・アドレス	オブジェクト・プログラム		ソース・プログラム
			ORG　9000H
9000_H	3E　FF	←	LD　A, 0FFH
9002_H	32　01　91	←	LD　(9101H), A
9005_H	C9	←	RET

このように６バイトのオブジェクト・プログラムができました．

これを実行するにはアドレス9000_Hからのメモリ上に，

```
3E      FF      32      01      91      C9
↓       ↓                               ↓
9000H   9001H ………………………………………………… 9005H
```

のような順に書き込みます．

ではMCモニタを使って，このプログラムをメモリに書き込み，実行して結果を確認してみましょう．

```
-S9000 ↵
9000 47 3E ↵
9001 F0 FF ↵
9002 A1 32 ↵      Sコマンドでアドレス9000Hからオブジェクト・プログラムを書き込む.
9003 D3 01 ↵
9004 10 91 ↵
9005 F3 C9 ↵
9006 3A .  ↵ ……「.」でSコマンドを終わる.

-D9000 9005 ↵ …………ダンプして確認.
9000  3E FF 32 01 91 C9
      書き込んだオブジェクト・プログラム.
-S9100 ↵
9100 E6 00 ↵
9101 01 00 ↵      結果がストアされるメモリを0クリアしておく(特に必要ではないが).
9102 28 00 ↵
9103 FA .  ↵ ……… Sコマンドを終わる.
-D9100 9102 ↵ ……… 0クリアの確認.
9100  00 00 00

-G9000 ↵ ………実習2のプログラムを実行する. スタートアドレスは9000H.

-D9100 9102 ↵ ………実行結果をダンプして確認.
9100  00 FF 00
         ↑
  アドレス9101HIに値FFHがストアされている.
-
```

Figure-8.2.3 実習2(1バイトのデータをメモリ上に書き込む)の実行

実習3 Aレジスタ以外のレジスタへのロード

ふたつの任意の1バイトデータをBレジスタとCレジスタにロードします．次にさきほどの「実習2」のプログラムを実行し，その後Bレジスタの内容をAレジスタに転送し，そのデータを任意のメモリにストアします．続いて同様にCレジスタの内容をAレジスタに転送し，任意のメモリにストアします．

ちょっと複雑になりましたが，このプログラムは使用していないレジスタ

にデータを一時的に保存しておき，必要になったときに取り出して使うという一例でもあります．

新しく登場する命令は，

「Bレジスタ⇦任意の1バイトデータ」の命令は，

LD B，●●……マシン語では「06 ●●」の2バイト．

「Cレジスタ⇦任意の1バイトデータ」の命令は，

LD C，●●……マシン語では「0E ●●」の2バイト．

「Aレジスタ⇦Bレジスタの内容」の命令は，

LD A，B ………マシン語では「78」の1バイト．

「Aレジスタ⇦Cレジスタの内容」の命令は，

LD A，C ………マシン語では「79」の1バイト．

の4種であり，その他の命令は実習1，2で使っているものです．

では具体的に，ふたつの1バイトデータを10_Hと11_Hとし，これらのデータをストアするアドレスを9200_H，9201_Hとして，この「実習3」のソース・プログラムを次に示します．

```
LD   B,10H      ………………Bレジスタに1バイトデータ「10H」をロード.
LD   C,11H      ………………Cレジスタに1バイトデータ「11H」をロード.
LD   A,0FFH     ┐
LD   (9101H),A  ┘実習2のプログラム*.
LD   A,B        ………………Bレジスタの内容をAレジスタに転送.
LD   (9200H),A  …………Aレジスタの内容をアドレス9200Hのメモリにストア.
LD   A,C        ………………Cレジスタの内容をAレジスタに転送.
LD   (9201H),A  …………Aレジスタの内容をアドレス9201Hのメモリにストア.
RET ……………………………当サブルーチンを終了してMCモニタに戻る.
```

このソース・プログラムのスタート・アドレスを今回は9050_Hとしてハンド・アセンブルしましょう．結果を次に示します．

*この間にどのような命令を挿入して実行しようと，BレジスタとCレジスタに新たに書き込みを行なわない限り，先にロードしたふたつのデータは保存される．

ロード・アドレス	オブジェクト・プログラム	ソース・プログラム
		ORG 9050H
9050_H	06 10 ←	LD B,10H
9052_H	0E 11 ←	LD C,11H
9054_H	3E FF ←	LD A,0FFH
9056_H	32 01 91 ←	LD (9101H),A
9059_H	78 ←	LD A,B
$905A_H$	32 00 92 ←	LD (9200H),A
$905D_H$	79 ←	LD A,C
$905E_H$	32 01 92 ←	LD (9201H),A
9061_H	C9 ←	RET

このように18バイトのオブジェクト・プログラムができました.

ではこれをMCモニタでスタート・アドレスの9050_Hからメモリに書き込み，実行して結果を確認してみましょう.

```
-S9050 ↵
9050 73 06 ↵
9051 19 10 ↵
9052 E9 0E ↵
9053 31 11 ↵
9054 00 3E ↵
9055 F0 FF ↵
9056 F5 32 ↵
9057 E5 01 ↵
9058 D5 91 ↵
9059 C5 78 ↵
905A CD 32 ↵
905B 6C 00 ↵
905C E8 92 ↵
905D 3A 79 ↵
905E 03 32 ↵
905F F0 01 ↵
9060 D3 92 ↵
9061 E4 C9 ↵
9062 C1 . ↵

-D9050 9061 ↵
9050   06 10 0E 11 3E FF 32 01 91 78 32 00 92 79 32 01
9060   92 C9
```

Sコマンドでオブジェクト・プログラムを書き込む. 今回のロード・アドレスは9050_Hから.

……Sコマンドを終わる.

……ダンプして確認.

```
-S9100↵……… 結果がストアされるメモリを0クリアしておく.
9100 E6 00↵
9101 01 00↵
9102 28 00↵
9103 FA . ↵

-S9200↵……… もうひとつのストアされるメモリを0クリアしておく.
9200 ED 00↵
9201 11 00↵
9202 40 00↵
9203 F0 . ↵

-G9050↵……… 実習3のプログラムを実行する.

-D9100 9102↵……… 実行結果をダンプして確認する.
9100  00 FF 00
          ~~ ……… 実習2と同じ結果.
-D9200 9202↵……… 実行結果をダンプして確認する.
9200  10 11 00
      ~~ ~~
      ↑  ↑
-   値10Hと11Hがストアされている.
```

Figure-8.2.4 実習3(Aレジスタ以外のレジスタへのロード)の実行

実習4 HLレジスタペアでアドレスされるメモリへのストア

Hレジスタと Lレジスタにすでにセットされている2バイトのデータ(16進4桁)をアドレス値とするメモリに,Aレジスタの内容をストアしてみましょう.

HLレジスタペアでアドレスされるメモリにAレジスタの内容をストアする命令は,

LD (HL), A………マシン語では「77」の1バイト.

という形式で表わします.( )の意味は,( )内のレジスタの値ではなく,その値をアドレスデータとした場合にアドレスされる「メモリの内容」であることを示しています.「実習2」を参照してください.

では具体的に，次のようなプログラムを作ってみましょう．

HLレジスタペアの2バイトが9250_Hを示すようにHレジスタに92_H，Lレジスタに50_Hをあらかじめロードしておき，Aレジスタにはメモリにストアするデータとして$7F_H$をロードしたあと，今回のメモリへのストア命令を実行します．この「実習4」のソース・プログラムを次に示します。

```
LD    H ,92H  ………………Hレジスタにデータ「92H」をロード(アドレス上位データ).*
LD    L ,50H  ………………Lレジスタにデータ「50H」をロード(アドレス下位データ).*
LD    A ,7FH  ………………Aレジスタにデータ「7FH」をロード(メモリにストアするデータ).
LD    (HL),A  ………………HLレジスタペアでアドレスされるメモリにAレジスタの内容をストア.
RET           ………………当サブルーチンを終了してMCモニタに戻る.
```

このソース・プログラムのスタート・アドレスを9000_Hとしてハンド・アセンブルしてみましょう．その結果を次に示します．

ロード・アドレス	オブジェクト・プログラム		ソース・プログラム
			ORG 9000H
9000_H	26 92	←	LD H , 92H
9002_H	2E 50	←	LD L , 50H
9004_H	3E 7F	←	LD A , 7 FH
9006_H	77	←	LD (HL) , A
9007_H	C9	←	RET

このように8バイトのオブジェクト・プログラムができました．

ではこれをMCモニタでメモリに書き込み，実行して結果を確認してみましょう．

*これらふたつの命令をひとつの命令にすることもできる．詳しくは8．9章を参照．

```
-S9000 ↲
9000 47 26↲   ┐
9001 F0 92↲   │
9002 A1 2E↲   │
9003 D3 50↲   │
9004 10 3E↲   ├ Sコマンドでオブジェクト・プログラムを書き込む.*
9005 F3 7F↲   │
9006 3A 77↲   │
9007 01 C9↲   ┘
9008 F0 .↲ ………Sコマンドを終わる.

-D9000 9007 ↲ ………ダンプして確認.
9000  26 92 2E 50 3E 7F 77 C9

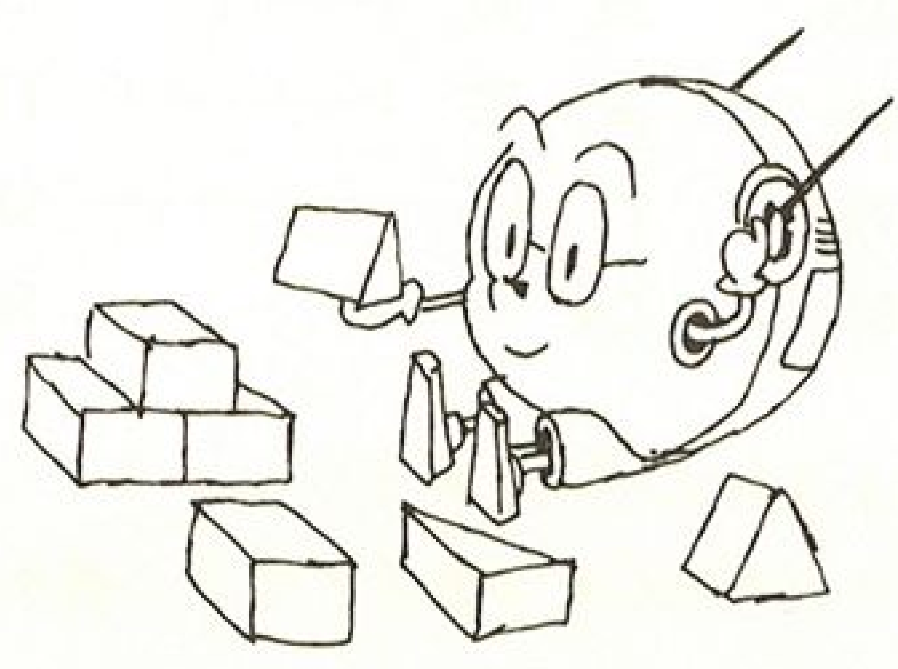
-S9250 ↲ ………結果がストアされるメモリを0クリアしておく.
9250 0E 00↲
9251 0A 00↲
9252 20 00↲
9253 AA . ↲

-G9000↲ ………実習4のプログラムを実行する.

-D9250 9252 ↲ ………実行結果をダンプして確認する.
9250  7F 00 00
       ↑
-   アドレス9250H に値 7FH がストアされている.
```

Figure-8.2.5　実習4（HLレジスタペアでアドレスされるメモリへのストア）の実行

実習1～4は8ビット・ロード命令のほんの一例です．命令表を見て，レジスタを替えてみたり，数値を変えてみたりして，各自でいろいろと試みてください．



*この実習を最後に，9000H 番台のエリアをフリーエリアとして使用できない場合に必要な変更箇所の指定は省略するので注意すること．

8.3

●8ビット算術演算命令

Aレジスタは，足し算や引き算，それに論理演算などいろいろな演算ができるレジスタで，「アキュムレータ」とも呼ばれています．その他のレジスタには「1を足す」，「1を引く」という機能はありますが，その他の演算機能はありません．

普通は，Aレジスタの中で演算が行なわれると考えていますが，内部的に厳密にいうと，実はAレジスタと演算を行なう部分は別のもので，実際に演算を行なうのはALU*(Arithmetic and Logic Unit)と呼ばれる演算部であり，その結果がAレジスタに残るということなのです．しかし私たちにはCPUの内部的な動きなど直接には関係ないので，あくまでAレジスタで演算が行なわれると考えてさしつかえありません．

このAレジスタが行なう各種の演算について，その主要なものを実習により解説していきましょう．



AレジスタとALUの働きに関する図解を次に示します．主要な8ビット算術演算命令の表は巻末をご覧ください．表の「ADC」命令や「SBC」命令は，本書では実習しませんが，多桁演算を行なう場合を考慮して，キャリー(C_Y)**を含めた加算，減算命令です．

*算術演算や論理演算を行なう論理回路ユニットのこと．コンピュータの処理機能の中枢になっている．
**キャリー・フラグ(C_Y)についての詳細は8．4章を参照．

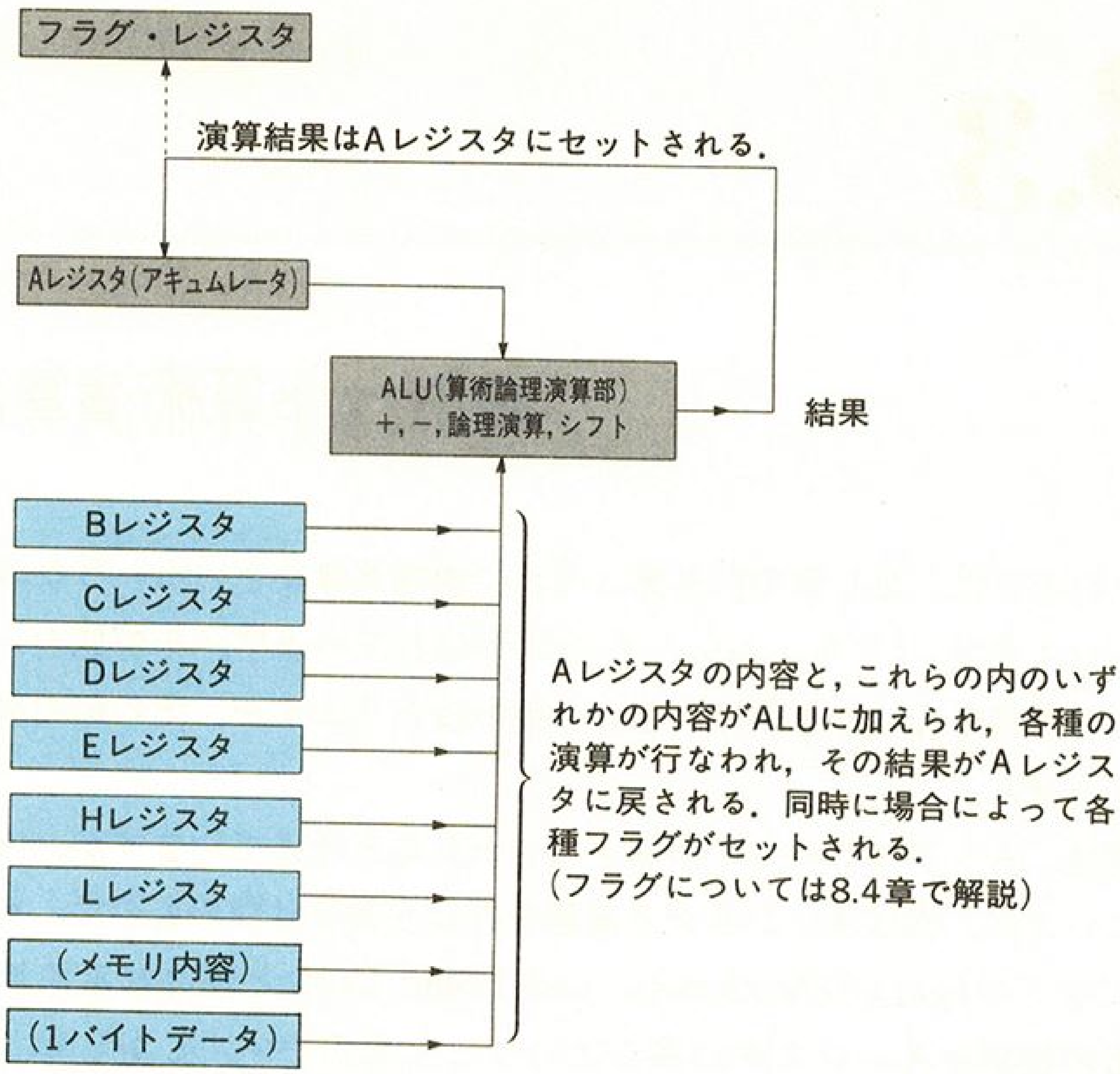


Figure-8.3.1　8ビット算術論理演算におけるAレジスタとALUの働き

実習5　8ビットデータの加減算(1)

Aレジスタと，1バイトデータとの加算および減算の実習です．Aレジスタの内容に任意の1バイトデータを加え，その結果を適当なアドレスのメモリにストアします．続けてその値から任意の1バイトデータを引き，その結果を次のアドレスにストアしてみましょう．

「Aレジスタの内容＋任意の1バイトデータ」の命令は，

ADD(アド) A，●●（1バイトデータ）………マシン語では「C6　●●」の2バイト.

「Aレジスタの内容－任意の1バイトデータ」の命令は，

SUB（サブトラクト） ●●（1バイトデータ） ………マシン語では「D6 ●●」の2バイト．

という形式で表わします．これらの演算結果はAレジスタにセットされます．

他の命令についてはすでに「実習1」その他で実習済なので「実習5」のソース・プログラムは次のようになります．最初のAレジスタの内容を10_H，加算する任意の1バイトデータを40_H，減算する1バイトデータを20_H，ストアするメモリ・アドレスを9100_Hと9101_Hにしておきましょう．

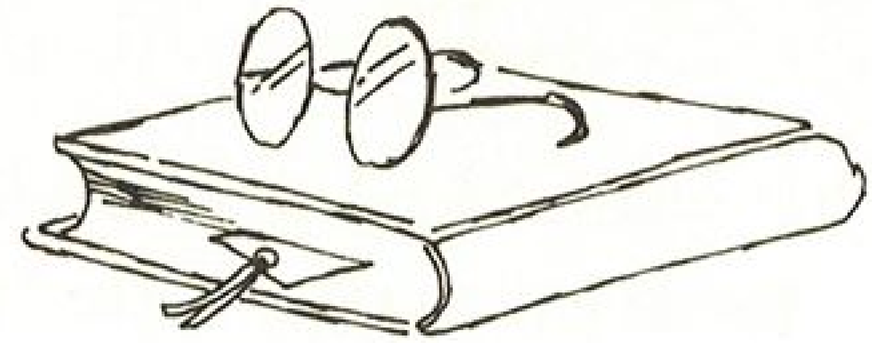
```
LD    A ,10H  ………… Aレジスタに1バイトデータ「10H」をロードする．
ADD   A ,40H  ………… Aレジスタの内容に1バイトデータ「40H」を加える．
LD    (9100H), A …… その結果がセットされているAレジスタの内容をア
                      ドレス9100Hのメモリにストアする．
SUB   20H ……………… Aレジスタの内容から1バイトデータ「20H」を引く．
LD    (9101H), A ……その結果を次のアドレスにストアする．
RET   ……………………… 当サブルーチンを終了してMCモニタに戻る．
```

上記ソース・プログラムをハンド・アセンブルして，結果を次に示します．スタート・アドレスは9000_Hとしてあります．

ロード・アドレス	オブジェクト・プログラム		ソース・プログラム
			ORG 9000H
9000_H	3E 10	←	LD A ,10H
9002_H	C6 40	←	ADD A ,40H
9004_H	32 00 91	←	LD (9100H), A
9007_H	D6 20	←	SUB 20H
9009_H	32 01 91	←	LD (9101H), A
$900C_H$	C9	←	RET

このように13バイトのオブジェクト・プログラムができました.

ではこれをMCモニタでメモリに書き込み，実行して結果を確認してみましょう.



```
-S9000↵
9000 47 3E↵
9001 F0 10↵
9002 A1 C6↵
9003 D3 40↵
9004 10 32↵
9005 F3 00↵     Sコマンドでオブジェクト・プログラムを書き込む.
9006 3A 91↵
9007 01 D6↵
9008 F0 20↵
9009 E6 32↵
900A FE 01↵
900B D3 91↵
900C 40 C9↵
900D F6 .↵  ……… Sコマンドを終わる.

-D9000 900C↵ ………ダンプして確認.
9000  3E 10 C6 40 32 00 91 D6 20 32 01 91 C9

-S9100↵ ………結果がストアされるメモリを0クリアしておく.
9100 E6 00↵
9101 01 00↵
9102 28 00↵
9103 FA .↵

-G9000↵ ………実習5のプログラムを実行する.

-D9100 9102↵ ………実行結果をダンプして確認する.
9100  50 30 00
       ↑   ↑
-  (10H+40H) (50H−20H)
```

Figure-8.3.2 実習5(8ビットデータの加減算(1))の実行

実習6　8ビットデータの加減算(2)

Aレジスタと各レジスタとの加算および減算の実習です．あらかじめセットしてあるAレジスタの内容をBレジスタに保存しておき，その値とCレジスタの内容を加算および減算して，その結果をメモリにストアしてみましょう．

「Aレジスタの内容＋各レジスタの内容」の命令は，各レジスタ名をrとすると，

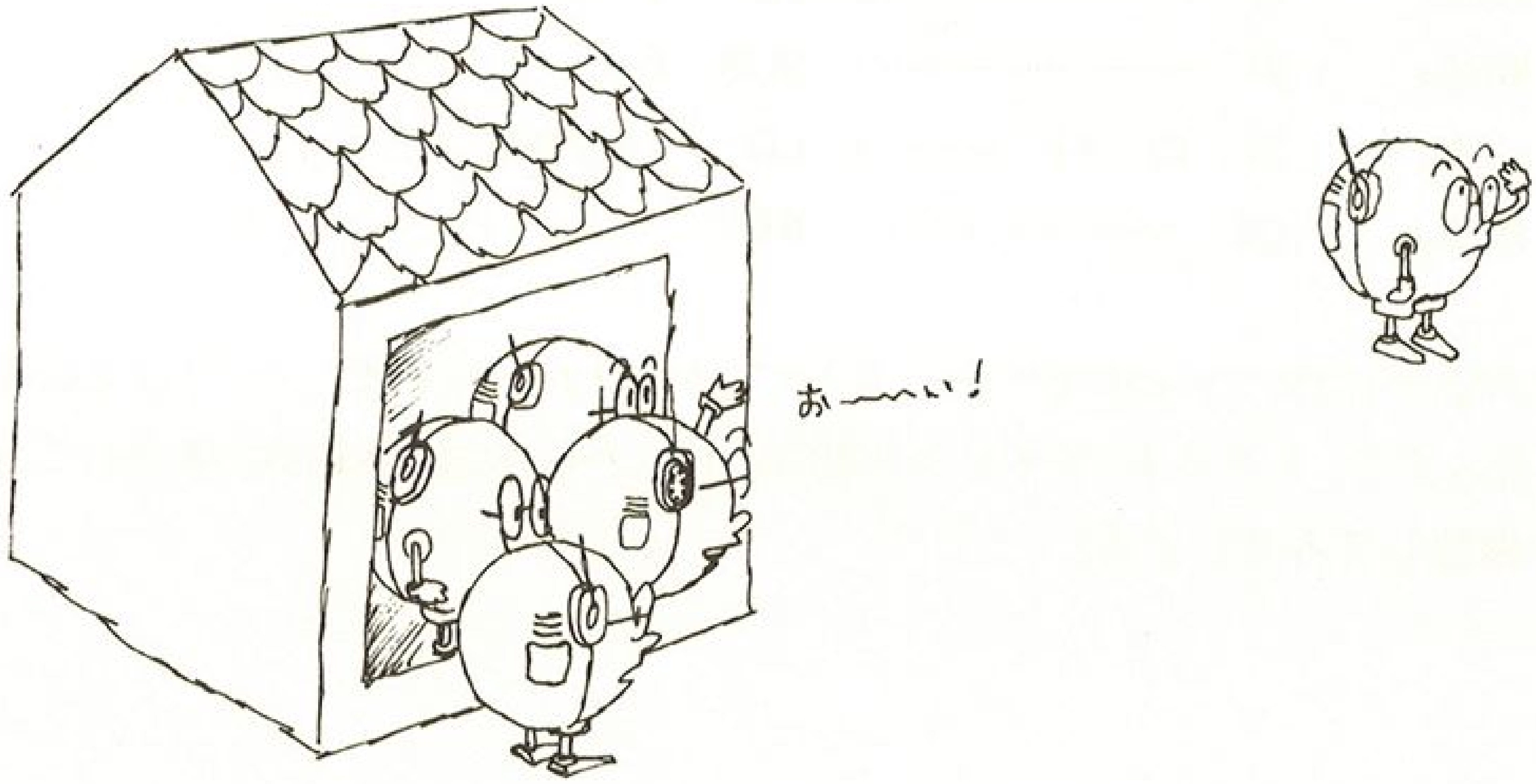
ADD（アド）　A，r …マシン語ではr＝Bの場合「80」，r＝Cの場合「81」，
　　　　　その他は表を参照．いずれも1バイトの命令．

「Aレジスタの内容－各レジスタの内容」の命令は，

SUB（サブトラクト）　r ………マシン語ではr＝Bの場合「90」，r＝Cの場合「91」，
　　　　　その他は表を参照．いずれも1バイトの命令．

という形式で表わします．この実行結果はいずれもAレジスタにセットされます．

よって「実習6」のソース・プログラムは次のようになります．あらかじめAレジスタには15_H，Cレジスタには03_Hをセットし，結果をストアするメモリのアドレスを9120_Hと9121_Hにしておきましょう．



LD	A , 15H	……………Aレジスタに1バイトデータ「15$_H$」をロード.
LD	B , A	………………Aレジスタの内容「15$_H$」をBレジスタに保存.
LD	C , 03H	……………Cレジスタに1バイトデータ「03$_H$」をロード.
ADD	A , C	………………Aレジスタの内容と，Cレジスタの内容を加算,
LD	(9120H) , A	………その結果がセットさているAレジスタの内容をアドレス9120$_H$のメモリにストア.
LD	A , B	………………Bレジスタに保存していた値「15$_H$」をAレジスタに転送.
SUB	C	……………………Aレジスタの内容からCレジスタの内容を引く.
LD	(9121H) , A	……その結果を次のアドレスにストア.
RET		……………………当サブルーチンを終了してMCモニタに戻る.

このソース・プログラムのスタート・アドレスを今回は9030$_H$としてハンド・アセンブルしましょう．その結果を次に示します．

ロード・アドレス	オブジェクト・プログラム	ソース・プログラム
		ORG 9030H
9030$_H$	3E 15 ←	LD A , 15H
9032$_H$	47 ←	LD B , A
9033$_H$	0E 03 ←	LD C , 03H
9035$_H$	81 ←	ADD A , C
9036$_H$	32 20 91 ←	LD (9120H) , A
9039$_H$	78 ←	LD A , B
903A$_H$	91 ←	SUB C
903B$_H$	32 21 91 ←	LD (9121H) , A
903E$_H$	C9 ←	RET

このように15バイトのオブジェクト・プログラムができました．ではこれをMCモニタで，スタート・アドレスの9030$_H$からメモリに書き込み，実行して結果を確認してみましょう．

```
-S9030↵
9030 FF 3E↵
9031 A7 15↵
9032 20 47↵
9033 06 0E↵
9034 21 03↵
9035 3D 81↵
9036 E8 32↵    Sコマンドでオブジェクト・プログラムを書き込む.
9037 CD 20↵
9038 7D 91↵
9039 E5 78↵
903A AF 91↵
903B 37 32↵
903C C9 21↵
903D 5B 91↵
903E 4D C9↵
903F 49 . ↵ ………Sコマンドを終わる.

-D9030 903E↵ ………ダンプして確認.
9030  3E 15 47 0E 03 81 32 20 91 78 91 32 21 91 C9

-S9120↵………結果がストアされるメモリを0クリアしておく.
9120 E9 00↵
9121 CD 00↵
9122 98 00↵
9123 E9 . ↵

-G9030↵ ………実習6のプログラムを実行する.

-D9120 9122↵………実行結果をダンプして確認する.
9120  18 12 00
       ↑   ↑
-   (15H+03H) (15H−03H)
```

Figure-8.3.3 実習6(8ビットデータの加減算(2))の実行

実習7 8ビットデータのインクリメント,デクリメント

各レジスタの値を+1,または−1する命令の実習です.+1または−1するには,実習5,6で行なった「ADD」,「SUB」を使えばもちろんできますが,+1と−1に関しては,それ専用の命令が用意されています.「INC(インクリメント)」

命令と「DEC(デクリメント)」命令がそれです．たとえば，

ADD　A,01　≒　INC A

であり，どちらもAレジスタの内容を＋１します．両者が異なる点は，ADDおよびSUB命令は１バイトまたは２バイトの命令で，Aレジスタのみが対象ですが，INCおよびDEC命令はかならず１バイトの命令で，その他のレジスタも対象となります．

また8.4章で解説するフラグの変化に関連する違いもあります．たとえばADDおよびSUB命令を実行すると，その演算結果に従いC_Y(キャリー)フラグが変化しますが，INCおよびDEC命令ではC_Yフラグは変化しません．

今回の実習は，Bレジスタに任意の１バイトデータをロードし，それをAレジスタに転送したあと，Aレジスタを－１します．そのAレジスタの内容をメモリにストアします．続けてBレジスタの内容を＋１し，それをAレジスタに転送後メモリにストアします．つまり最初の１バイトデータの－１と＋１した値がメモリにストアされるわけです．

新しく登場する命令は次の２種類です．

レジスタの内容を＋１する命令は，各レジスタ名をrとすると，

INC(インクリメント)　r………マシン語ではr＝Aの場合「3C」，r＝Bの場合「04」，その他は表を参照．いずれも１バイトの命令．

この命令はAレジスタに関係なく，それぞれのレジスタ自身が＋１されます．

レジスタの内容を－１する命令は，各レジスタ名をrとすると，

DEC(デクリメント)　r………マシン語ではr＝Aの場合「3D」，r＝Bの場合「05」，その他は表を参照．いずれも１バイトの命令．

この命令もAレジスタに関係なく，それぞれのレジスタ自身が－１されます．

よって「実習７」のソース・プログラムは次のようになります．１バイトデータを10_H，ストアするメモリ・アドレスは最初のものを9200_H，次のものを9201_Hとしておきましょう．

```
LD    B ,10H   ………… Bレジスタに１バイトデータ「10H」をロード.
LD    A ,B     ……………… それをAレジスタに転送.
DEC   A        ………………… Aレジスタの内容(現在は10H)を－１.
LD    (9200H) ,A  …… その結果をアドレス9200Hのメモリにストア.
INC   B        ………………… Bレジスタの内容(現在は10H)を＋１.
LD    A ,B     ……………… それをAレジスタに転送.
LD    (9201H) ,A  …… その内容をアドレス9201Hのメモリにストア.
RET            ………………… 当サブルーチンを終了してMCモニタに戻る.
```

このソース・プログラムのスタート・アドレスを今回は9500_Hとしてハンド・アセンブルしましょう．その結果を次に示します．

ロード・アドレス	オブジェクト・プログラム	ソース・プログラム
		ORG 9500H
9500_H	06 10	LD B ,10H
9502_H	78	LD A ,B
9503_H	3D	DEC A
9504_H	32 00 92	LD (9200H) ,A
9507_H	04	INC B
9508_H	78	LD A ,B
9509_H	32 01 92	LD (9201H) ,A
$950C_H$	C9	RET

このように13バイトのオブジェクト・プログラムができました．ではこれをMCモニタで，スタート・アドレスの9500_Hからメモリに書き込み，実行して結果を確認してみましょう．

```
-S9500↵
9500 00 06↵
9501 00 10↵
9502 00 78↵
9503 00 3D↵
9504 00 32↵
9505 04 00↵
9506 01 92↵        Sコマンドでオブジェクト・プログラムを書き込む.
9507 1A 04↵
9508 0E 78↵
9509 FF 32↵
950A 04 01↵
950B 00 92↵
950C 1A C9↵
950D 07 . ↵  ………Sコマンドを終わる.

-D9500 950C↵………ダンプして確認.
9500   06 10 78 3D 32 00 92 04 78 32 01 92 C9

-S9200↵………結果がストアされるメモリを0クリアしておく.
9200 ED 00↵
9201 11 00↵
9202 40 00↵
9203 F0 . ↵

-G9500↵………実習7のプログラムを実行する.

-D9200 9202↵………実行結果をダンプして確認する.
9200   0F 11 00
-       ↑  ↑
 (10H－1H) (10H＋1H)
```

Figure-8.3.4　実習7(8ビットデータのインクリメント，デクリメント)の実行

実習8　8ビットデータの論理演算

Aレジスタと1バイトデータ間，およびAレジスタと各レジスタ間でAND(アンド)とOR(オア)をとる実習です．実習にはいる前に「AND」と「OR」の解説をしておきましょう．

AND, ORなどは論理演算と呼ばれており，その名の通り論理的(ロジカル)に演算を行なうものです（1か0，つまりあるかないかを問題にする)．この論理演算と

普通の演算（算術演算）との違いをここで説明しておきましょう．

算術和（＋）
0 ＋ 0 ＝ 0
0 ＋ 1 ＝ 1
1 ＋ 0 ＝ 1
1 ＋ 1 ＝ 2

OR 論理和（∨）
0 ∨ 0 ＝ 0
0 ∨ 1 ＝ 1
1 ∨ 0 ＝ 1
1 ∨ 1 ＝ 1

算術積（×）
0 × 0 ＝ 0
0 × 1 ＝ 0
1 × 0 ＝ 0
1 × 1 ＝ 1

AND 論理積（∧）
0 ∧ 0 ＝ 0
0 ∧ 1 ＝ 0
1 ∧ 0 ＝ 0
1 ∧ 1 ＝ 1

Figure-8.3.5　ANDとOR

1ビット単位での組合わせによる算術および論理和・積の表を上に示しました．このなかで，算術和の「1＋1＝2」と，論理和(OR)の「1∨1＝1」に注目してください．ビット単位の論理和あるいは論理積などは，「あ[1]る」か「な[0]い」かのふたつの状態のみを問題にしています．算術和の場合のように，演算結果の値そのものを問題にするわけではありません．よって，「あ[1]る」と「あ[1]る」との和は「あ[1]る」となるわけです．

しばしば使われるものに，「XOR（エクスクルーシブ・オア）」という論理演算があります．参考までにこの表も示しておきます．

XOR 排他的論理和（⊻）
0 ⊻ 0 ＝ 0
0 ⊻ 1 ＝ 1
1 ⊻ 0 ＝ 1
1 ⊻ 1 ＝ 0

Figure-8.3.6　XOR（排他的論理和）

実際のAND命令やOR命令などの論理演算命令は，1バイト単位でその8ビットのそれぞれについて演算します．その具体的な例を次に示します．

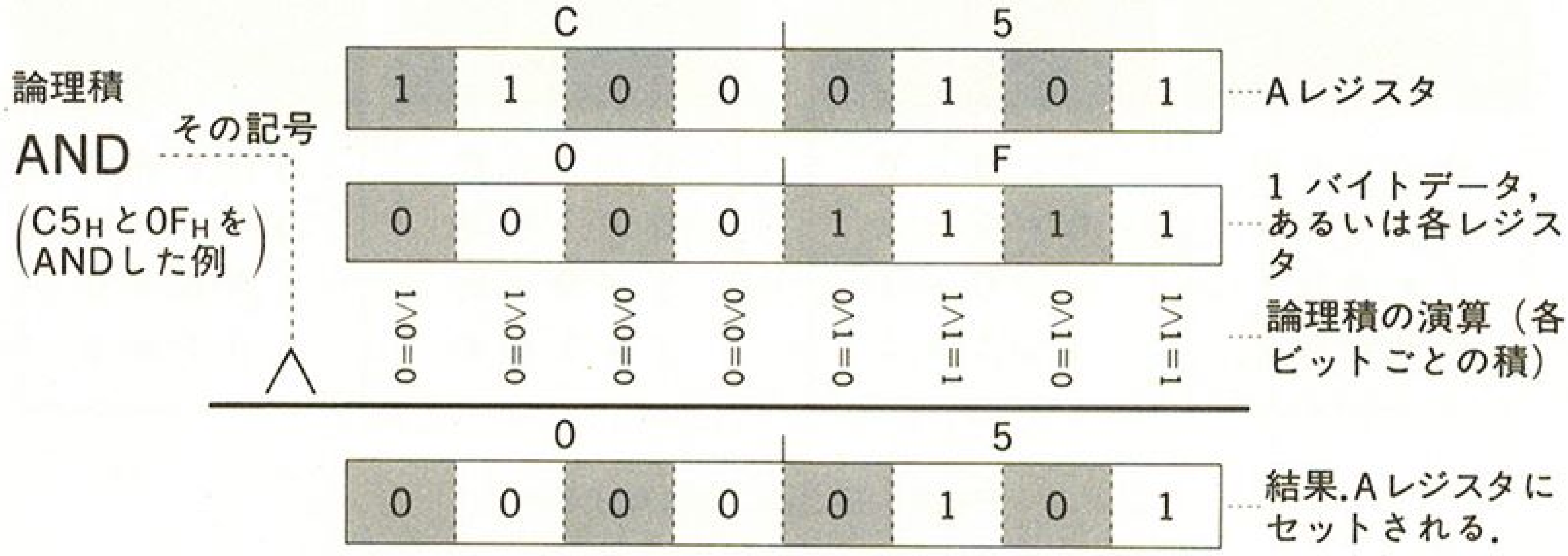


Figure-8.3.7　AND命令の図解

ANDの利用の一例として，このように「$0F_H$」をANDすると，上位4ビットはすべて0になり，下位の4ビットはもとのまま残ることになります．このようなとき「上位4ビットをマスクした」と表現します．

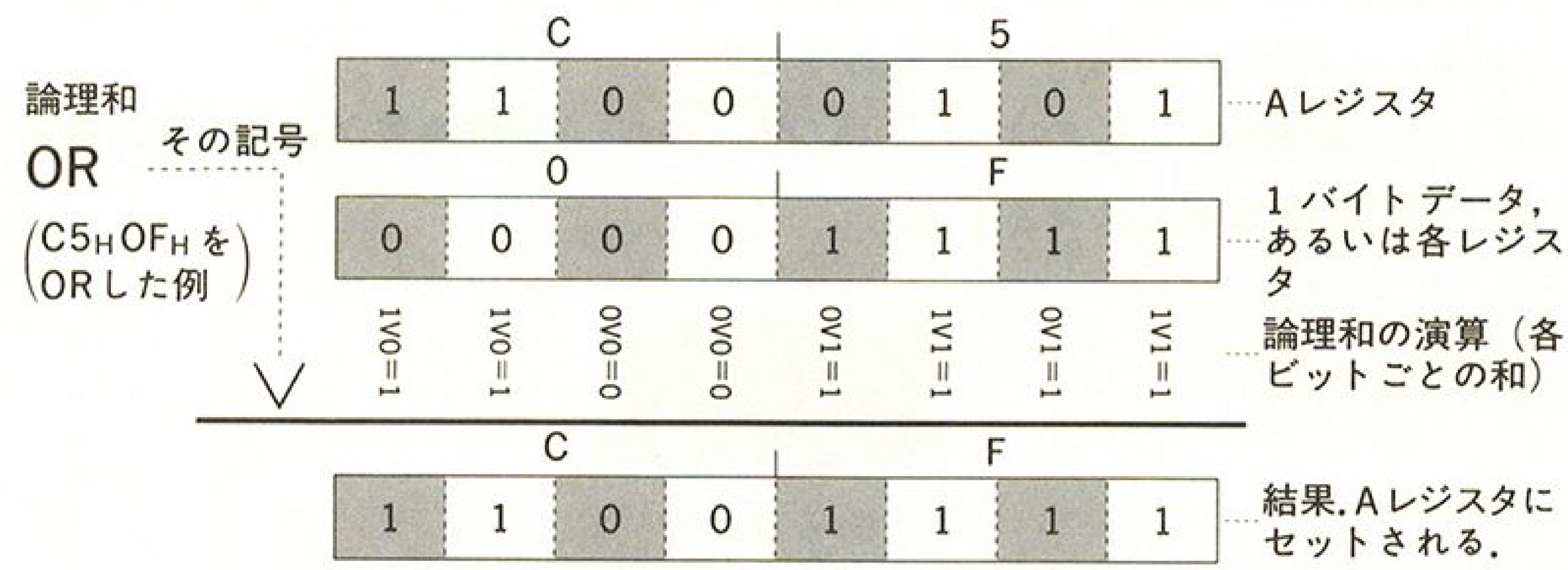


Figure-8.3.8　OR命令の図解

ORの利用の一例として，このように「$0F_H$」をORすると，上位4ビットはもとのままで，下位4ビットはすべて1になります．つまり下位4ビットを1でマスクしたことになります．

AND命令は，Aレジスタの内容と1バイトデータをANDする場合は，

AND ●● ………マシン語では「E6 ●●」の2バイトの命令.
1バイトデータ

また，Aレジスタの内容と各レジスタをANDする場合は，各レジスタ名をrとすると，

AND r ………マシン語ではr=Bの場合「A0」，r=Cの場合「A1」，その他は表を参照．いずれも1バイトの命令.

という形式で表わします.

次にOR命令は，Aレジスタの内容と1バイトデータをORする場合は，

OR ●● ………マシン語では「F6 ●●」の2バイトの命令.
1バイトデータ

また，Aレジスタの内容と各レジスタをORする場合は，各レジスタ名をrとすると，

OR r ………マシン語ではr=Bの場合「B0」，r=Cの場合「B1」，その他は表を参照．いずれも1バイトの命令.

という形式で表わします.

「実習8」は次のようなプログラムにしましょう．Figure-8.3.7およびFigure-8.3.8の例に合わせて，まずAレジスタには$C5_H$をセットしておきます．これとANDする1バイトの値として$0F_H$，ORするBレジスタの値として$0F_H$をセットします．そしてANDとOR命令を実行して，その結果をアドレス9300_Hと9301_Hのメモリにストアします．結果はFigure-8.3.7と8.3.8のとおりになるでしょうか.



このソース・プログラムは次のようになります.

```
LD     A , 0C5H ………1バイトデータ「C5H」をAレジスタにロード.
AND    0FH ……………それと1バイトデータ「0FH」をAND.
LD     (9300H), A …… 結果がセットされているAレジスタの内容をアドレ
                      ス9300Hのメモリにストア.
LD     A , 0C5H ……もう一度Aレジスタに「C5H」をロード.
LD     B , 0FH ………Bレジスタには「0FH」をロード.
OR     B…………………Aレジスタと BレジスタをORする.
LD     (9301H), A ……その結果をアドレス9301Hのメモリにストア.
RET    ……………………当サブルーチンを終了してMCモニタに戻る.
```

このソース・プログラムのスタート・アドレスを9000_Hとしてハンド・アセンブルし，その結果を次に示します.

ロード・アドレス	オブジェクト・プログラム		ソース・プログラム
			ORG 9000H
9000_H	3E C5	←	LD A , 0C5H*
9002_H	E6 0F	←	AND 0FH
9004_H	32 00 93	←	LD (9300H) , A
9007_H	3E C5	←	LD A , 0C5H
9009_H	06 0F	←	LD B , 0FH
$900B_H$	B0	←	OR B
$900C_H$	32 01 93	←	LD (9301H) , A
$900F_H$	C9	←	RET

このように16バイトのオブジェクト・プログラムができました. ではこれをMCモニタで，スタート・アドレスの9000_Hからメモリに書き込み，実行して結果を確認してみましょう.

*アセンブラでプログラムを組む場合，A～Fで始まる16進数の数値の先頭には，「0」を付ける必要がある. 6.2章を参照.

```
-S9000↲
9000 47 3E↲
9001 F0 C5↲
9002 A1 E6↲
9003 D3 0F↲
9004 10 32↲
9005 F3 00↲
9006 3A 93↲
9007 01 3E↲    Sコマンドでオブジェクト・プログラムを書き込む.
9008 F0 C5↲
9009 E6 06↲
900A FE 0F↲
900B D3 B0↲
900C 40 32↲
900D F6 01↲
900E 01 93↲
900F D3 C9↲
9010 40 . ↲ ………Sコマンドを終わる.

-D9000 900F↲………ダンプして確認.
9000  3E C5 E6 0F 32 00 93 3E C5 06 0F B0 32 01 93 C9

-S9300 ↲………結果がストアされるメモリを0クリアしておく.
9300 42 00↲
9301 F0 00↲
9302 32 00↲
9303 02 . ↲

-G9000↲ ………実習8のプログラムを実行する.

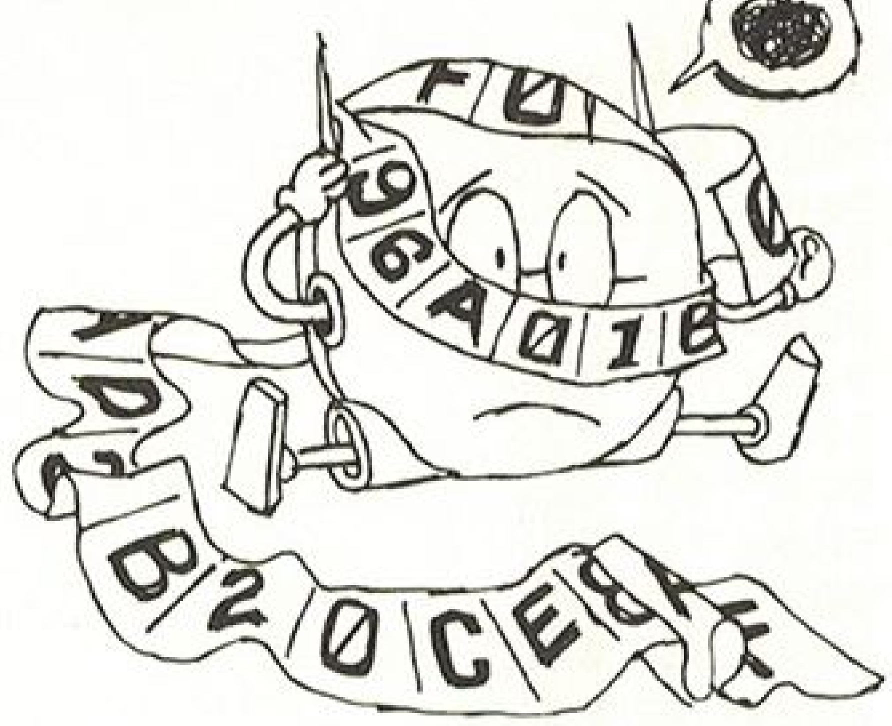
-D9300 9302 ↲………実行結果をダンプして確認する.
9300  05 CF 00
      ~~ ~~
      ↑  ↑
-     本文の図解と同じ結果.
```

Figure-8.3.9　実習8(8ビットデータの論理演算)の実行

論理演算は，特定のビットを1にしたり0にしたりする場合などにも使用しますが，そのような場合は「C5」というような16進の値より「11000101」というビットパターンでものを考えた方が便利です.

よって本格的なアセンブラはそのソース・プログラムを書くときに,数値を10進や16進だけでなく，ビットパターン(つまり2進)でも表現できるように

なっています．参考までにこの「実習8」のプログラムをビットパターンによる数値表現でアセンブルした例を示しておきます．



```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

9000  3EC5    0001        ORG    9000H
              0002 ;
9000  3EC5    0003        LD     A,11000101B
9002  E60F    0004        AND    00001111B
9004  320093  0005        LD     (9300H),A
              0006 ;
9007  3EC5    0007        LD     A,0C5H
9009  060F    0008        LD     B,00001111B
900B  B0      0009        OR     B
900C  320193  0010        LD     (9301H),A
900F  C9      0011        RET

Errors        0
```

ロード・アドレス　オブジェクト・プログラム　ラインNo.　ソース・プログラム

$C5_H$のビットパターン．

$0F_H$のビットパターン．

16進数には最後に「H」を付けるように，ビットパターンでは最後に「B」を付ける．

Figure-8.3.10　ビット・パターンによる数値表現のアセンブル例

●● 比較(コンペア)命令 ●

Aレジスタの内容と任意の1バイトデータとの比較(コンペア)，あるいはAレジスタの内容と各レジスタの内容との比較を行なう命令がCP(コンペア)命令です．具体的に

はALUで，

　　(Aレジスタの内容)−(1バイトデータ)
　あるいは
　　(Aレジスタの内容)−(各レジスタの内容)

の演算を行ない，その結果が0であったか，負であったかなどいくつかの「演算結果の状態」をフラグ・レジスタの該当ビットにセットします．これらの演算はALU内で行なわれますが，その演算結果は今までの演算命令のようにAレジスタにセットされません．よってこのコンペア命令を実行してもAレジスタの内容は変化せず，もとのままです．つまりコンペア命令は各種フラグをセットするだけの命令であるわけです．

　コンペア命令は，1バイトデータとの比較の場合，

　　CP（コンペア）　●●（1バイトデータ）………マシン語では「FE　●●」の2バイトの命令．

　各レジスタとの比較の場合は，各レジスタ名をrとすると，

　　CP　r …………………マシン語ではr＝Bの場合「B8」，r＝Cの場合「B9」，その他は表を参照．いずれも1バイトの命令．

という形式で表わします．

　これらのコンペア命令を使った実習は，条件分岐命令と切り離すことができないため，8．5章の「ジャンプ命令」，8．6章の「コール，リターン命令」の中の条件分岐の実習10，11のいくつかで当コンペア命令を取り扱います．

　また，次の8．4章で各種フラグと条件分岐についての概要を解説していますので，8．5〜6章を参照する前にはかならず8．4章に目を通しておいてください．

8.4

●各種フラグと条件分岐

各種のフラグと条件分岐(ジャンプ，コール，リターン)は別々に解説することはできません．コンピュータの重要な機能のひとつに「判断」がありますが，この判断，すなわち「××ならば○○する」という機能こそフラグによる条件分岐にほかならないのです．本節ではこの「フラグ」と「条件分岐」の概要を解説します．また個々の命令の具体例はこれ以後の実習プログラムで解説していますので参照してください．

●● フラグ ●

フラグについてはFigure-7.2.1とそれに続く本文でも紹介されていますが，ここでさらに詳しく解説しましょう．

「フラグ」は算術演算，論理演算，のちほど解説するローテートやシフトなどの命令を実行したときに，その結果の状態(答ではない)を知らせるためにセットされる「旗」のことです．私たちはこの旗の状態を見てそのあとの行動を決定し，プログラムを進行させます．

フラグはFigure-7.2.1に示されているフラグ・レジスタにセットされます．ただしフラグ・レジスタは他のレジスタのように8ビット単位で動作をするわけではなく，S(サイン)，Z(ゼロ)，C_Y(キャリー)などの個々のフラグごと，つまり1ビット単位で単独に動作します．よってこのフラグ・レジスタだけでは，他のレジスタのように8ビット・レジスタとしてプログラム上から自由に使用することはできま

せん.「LD　A，F」とか「ADD　A，F」(フラグ・レジスタ名をたとえば「F」として)などという命令はいっさいありませんので注意してください.

フラグの種類はZ-80CPUの場合，6種類がありますが，本書ではその内，最も多く使用され最も重要な「Z（ゼロ）」と「C_Y（キャリー）」フラグについて実習解説を行ないます.

フラグ・レジスタと各フラグの機能を次に示しておきます．また，各種演算命令が実行されたときのゼロ・フラグとキャリー・フラグの変化の様子は，巻末の機能別主要命令表に示されていますので必要なときに参照してください.

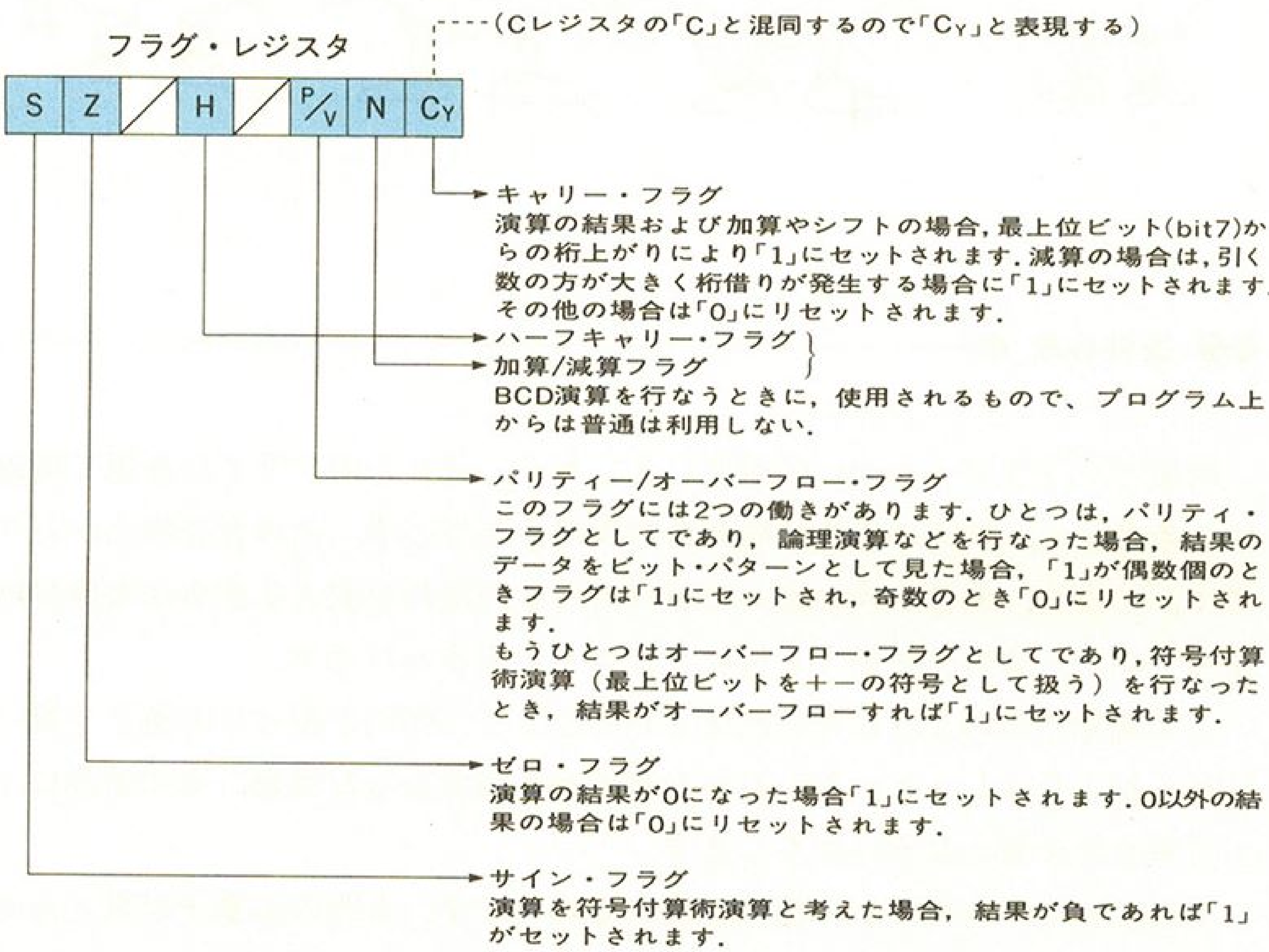
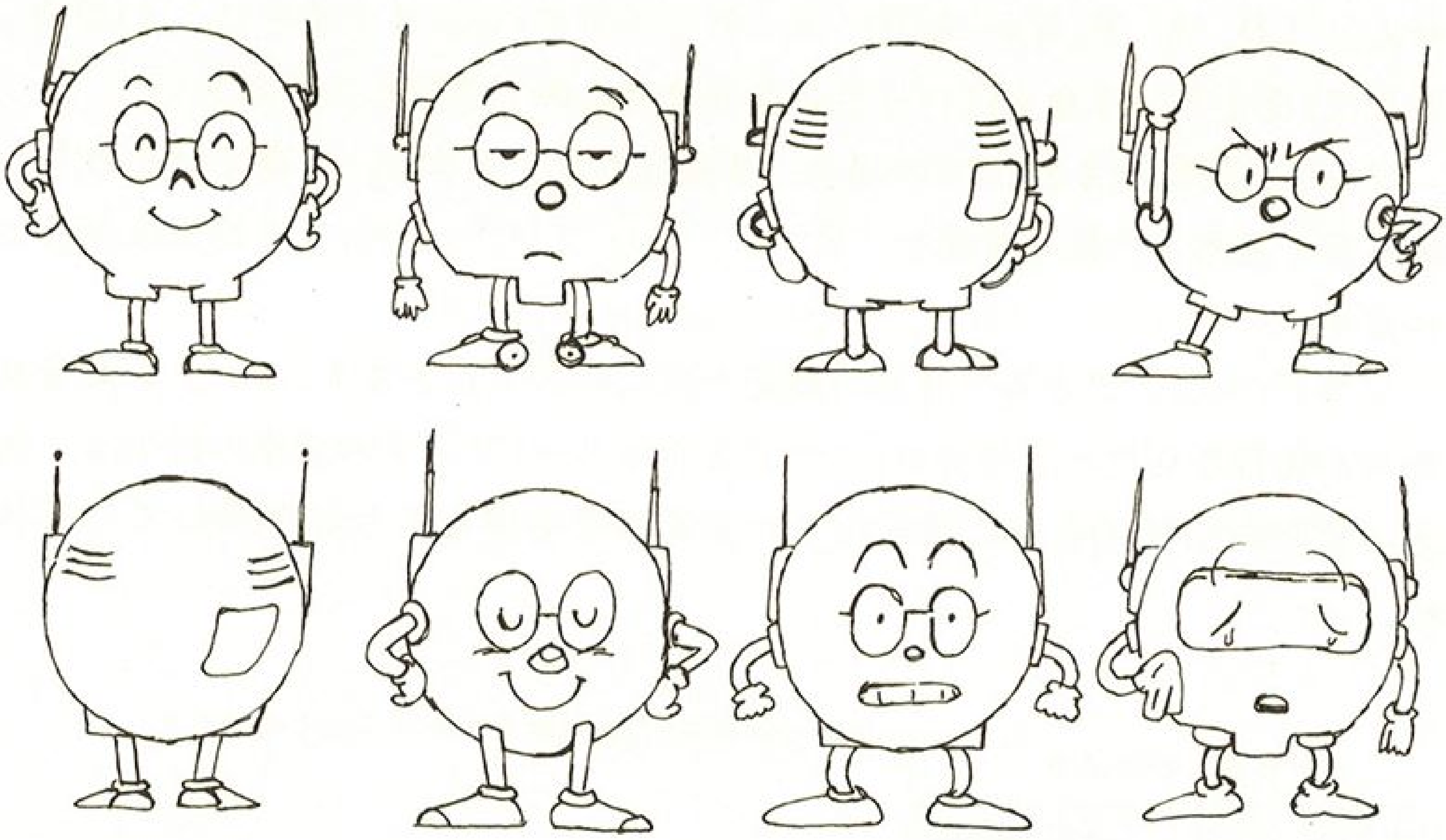


Figure-8.4.1　フラグ・レジスタと各フラグの機能



●● 条件分岐 ●

前項で「フラグ」について解説しましたが，それらのフラグは当項で解説する条件ジャンプ等の条件分岐命令で利用されてこそ，その存在理由があります．つまり，ある条件によってプログラムの流れを変える命令が条件分岐命令であり，条件を判断する材料がフラグであるわけです．

この関係は次のように考えればよいでしょう．100円を持ってお菓子を買いに行ったとしましょう．買いたいお菓子がａ円であった場合，その値段によって買えたり買えなかったりします．

それによりプログラムの流れが変わるわけです．ａ円のお菓子が買えるか買えないかの判断は手持ちの100円とａ円とを比較すれば簡単にわかります．比較の命令は次の節で実習しますが，「CP（コンペア）　●●」または「CP　r」という命令です．いずれもＡレジスタの内容と１バイトデータ，または各レジスタ

の内容とを比較し，同じ値ならZ(ゼロ)フラグを「1」にセットし，Aレジスタの内容の方が小さければC_Y(キャリー)フラグを「1」にセットします．

つまりコンペア命令を実行すれば，お菓子が買えるか買えないかがわかるわけです(減算命令「SUB」を使っても判断することができます．後述)．いや，本当はまだわかりません．CP命令を実行しただけでは，その結果により各フラグがセットされているだけで，私たちにはまだわかっていません．その次の命令で必要なフラグを見にいって初めてその結果がわかるのです．

必要なフラグを見にいき，その結果によりプログラムの流れを変えるのが条件分岐命令なのです．つまり条件ジャンプ／コール／リターン命令は，「フラグを見にいく」仕事と，「それによりプログラムの流れを変えたり変えなかったりする」仕事のふたつを同時に行なうわけです．

話をお菓子に戻すと，コンペア命令を実行して，買えるか買えないかを判断できるフラグがセットされたあとは，条件分岐命令を実行して自分の思うプログラムに導けばよいわけです．その導き方にジャンプ命令で分岐するものと，コール命令で分岐するものとがありますが，これはプログラマーが自由に使い分ければよい問題です．ここではジャンプ命令による分岐を行なうとして，このお菓子の話をプログラムにしてみました．それを次に示します．

(ロード)
LD A,100 ………… 100(円)をAレジスタにロード．

(コンペア)
CP ●● ………… Aレジスタの内容(100)と●●(円)を比較(100−●●)．100−●●＜0ならばキャリー・フラグ(C_Y)に「1」が立つ．

(ジャンプ if キャリー)
JP C,[買えない] ………… JP：ジャンプせよ——C：もしキャリー・フラグが「1」であれば——[買えない]のアドレスへ．

[買える] ………… もしキャリー・フラグが「1」でなかったらジャンプせずにここへ来る(●●が●●≦100であればここへ来る)．

買える場合のプログラム
〜

[買えない] ………… ●●が●●＞100であれば条件ジャンプにより，ここにジャンプして来る．

買えない場合のプログラム
〜

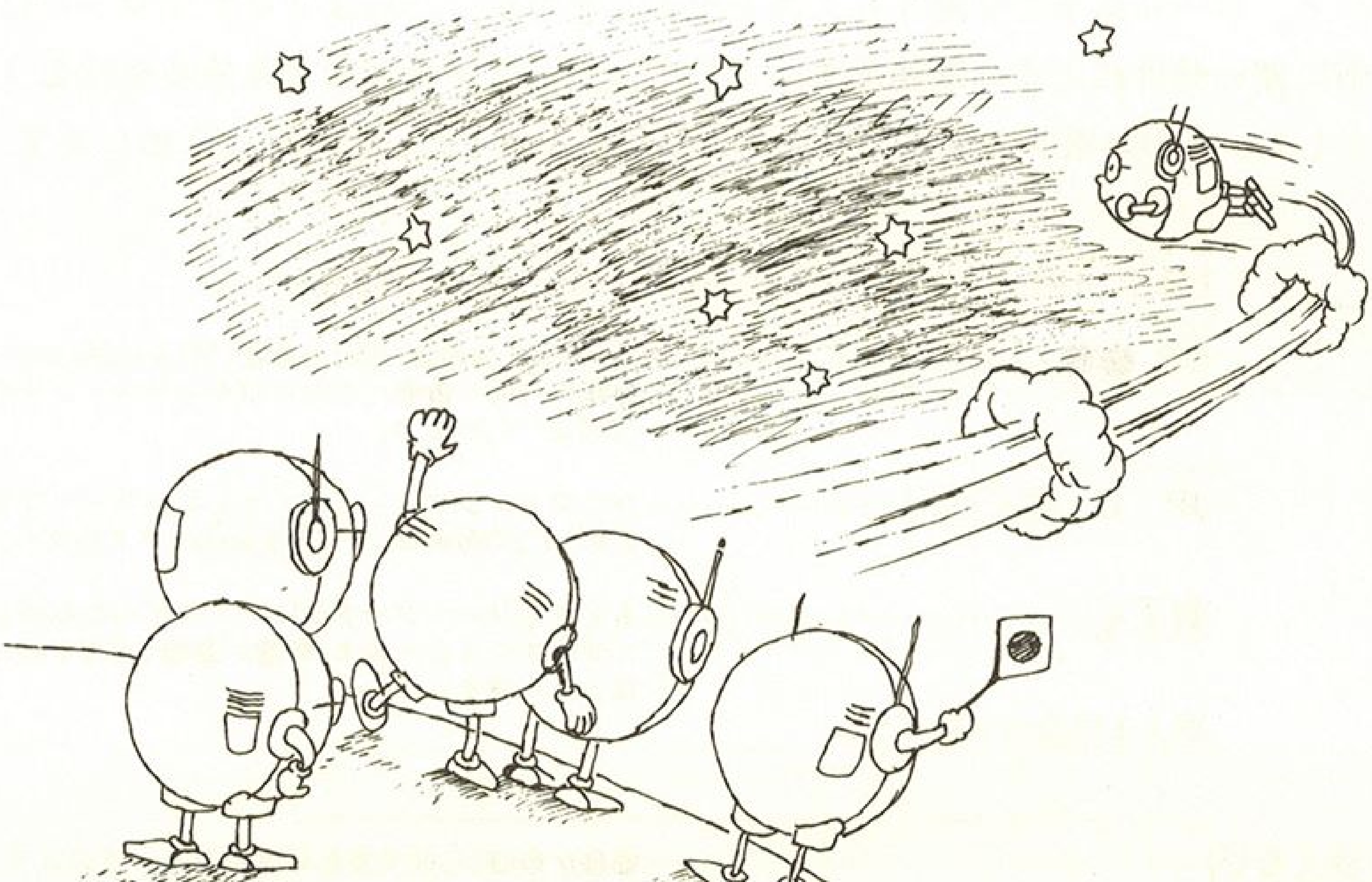
Figure-8.4.2 キャリーフラグによる条件ジャンプ

これはキャリー・フラグを利用した条件ジャンプの一例ですが，この例から「フラグ」と「条件分岐」の概念が一応理解できるのではないかと思います．さらに詳しくは，以後の各命令の実習において解説します．

上記のプログラム中の 買えない という表示は，アセンブラ用語で「ラベル」というものです．アセンブラのソース・プログラムでは，ジャンプ先のアドレスや，コール先のアドレスを16進の数値として表わす必要はなく，いくつかのアルファベットと数字を適当に組み合わせた「名札(ラベル)」で表わすことができます．ソース・プログラムの段階では具体的な数値アドレスを持たないラベルでも，アセンブルすることにより自動的に数値化され，オブジェクト・プログラムに組み込まれます．

また，ラベルは「シンボル」とも呼ばれ，アドレスを表わすために使われるだけではなく，２バイトまでの数値を表わすものとしても使用されます．

ラベルの具体的な使用例としては，Figure-8.5.4のアセンブルリストを参照してください．



8.5

●ジャンプ命令

コンピュータのプログラムはメモリのアドレスの低い方から高い方へ向って順に格納されており，それをCPUは内部のプログラム・カウンタに従って順に読み出してはCPUに取り込んで解析し実行していきます．このことは7.1章「プログラム実行のメカニズム」で詳しく解説しました．

つまりプログラムの流れはPCと呼ばれるプログラム・カウンタがコントロールしており，このPCを強制的に操作しない限り流れはアドレスの下位から上位へ順次進んでいくだけなのです．

この，プログラム・カウンタを強制的に操作して，プログラムの流れを変える方法は次の4つがあります．

1. ジャンプ命令
2. コール命令
3. リターン命令
4. 割込み（インタラプト）……これはプログラム上の命令（インストラクション）ではなく，CPUをハードウエア的にコントロールする方法．

いずれもコンピュータのプログラミング上なくてはならない重要な命令であり，これらがなくてはほとんどのプログラムは組むことができないでしょう．

本節では1のジャンプ命令の実習解説を行ない，次の節で2と3のコールとリターン命令の実習解説を行ないます．4の割込みに関しては本書では扱っていません．

ジャンプ命令には，「無条件ジャンプ命令」と「条件ジャンプ命令」とがありますが，いずれもプログラム・カウンタを任意のアドレス値にセットする命令です．ジャンプ命令を実行すると同時に指定したアドレスにシャンプし，そこからプログラムの実行を続行します．ひとつ注意しておくことは，ジャンプ命令を実行し，任意のアドレスへジャンプしてしまえば，「いったい私はどこからジャンプして来たのか」ということはいっさいわからなくなります．つまりジャンプさえすればこの命令の役目はすべて終了です．この点が同じ分岐命令でも次の節のコール，リターン命令と大きく異なるところです．

主要ジャンプ命令の表は，巻末にまとめて示してありますので参照してください．

無条件ジャンプ命令は，

JP　○○●●………マシン語では「C3　●●　○○」の3バイトの命令．

（○○●●：2バイトのメモリ・アドレス値）

条件ジャンプ命令は，各フラグ条件を×とすると，

JP　×，○○●●………マシン語では×＝C(つまりC_Y（キャリー）フラグが「1」でジャンプ)の場合「DA　●●　○○」，×＝Z(つまりZ（ゼロ）フラグが「1」でジャンプ)の場合「CA　●●　○○」の3バイトの命令．

という形式で表わします．

「JP　C，○○●●」は，C_Y（キャリー）フラグが「1」にセットされていれば，アドレス○○●●へジャンプせよ，という命令であり，「JP　Z，○○●●」は，Z（ゼロ）フラグが「1」にセットされていれば，アドレス○○●●へジャンプせよ，という命令です．これらはいずれも「××ならば△△」という命令ですが，この逆の「××でないならば△△」という命令も用意されています．それらの命令を次に示します．

JP NC（ノンキャリー），○○●●………マシン語では「D2　●●　○○」の3バイトの命令.

JP NZ（ノンゼロ），○○●●………マシン語では「C2　●●　○○」の3バイトの命令.

このように逆の条件で実行される命令が用意されているのはプログラミング技法上たいへん便利ですが，ただ一文字「N」が付くか付かないかで，動作はまったく逆になりますので十分注意して使ってください.

実習9　無条件ジャンプ

無条件ジャンプの実習です．実習6（プログラムのスタート・アドレス＝9030H）と実習7（スタート・アドレス＝9500H）を利用して，次のようにジャンプ命令を実習してみましょう.

この一連のプログラムは次のようにメモリ上に配置されます.

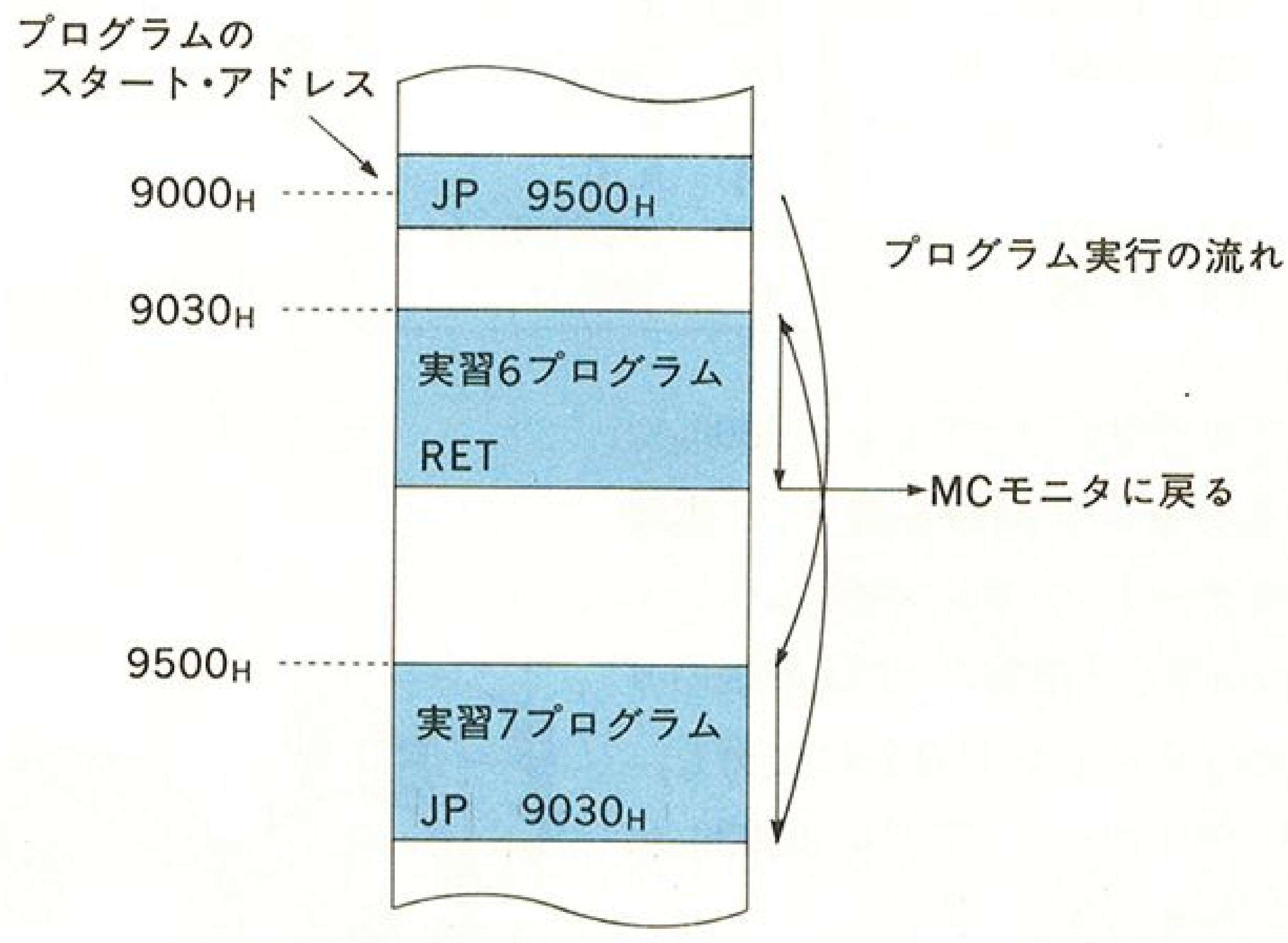


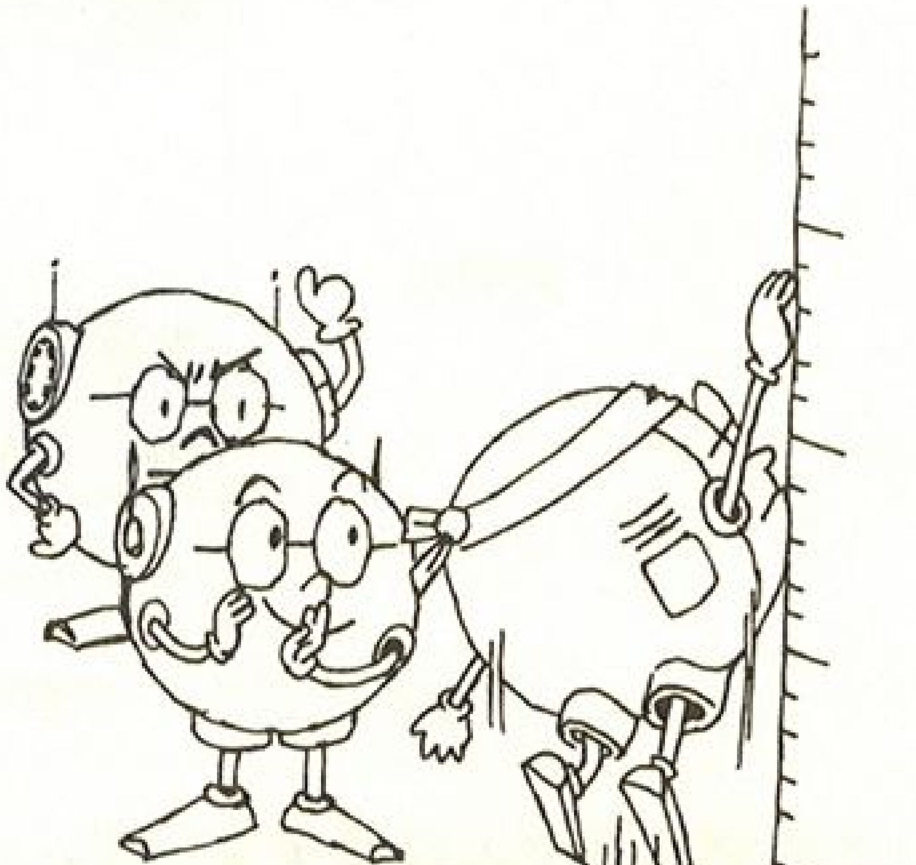
Figure-8.5.1　実習9のプログラム実行の流れ

このプログラムは簡単なので，ハンド・アセンブルした結果を次に示します.

ロード・アドレス	オブジェクト・プログラム	ソース・プログラム		
		ORG　9000H		
9000_H	C3 00 95	JP　9500H		アドレス9500_Hにジャンプ.
9030_H	3E 15	LD　A, 15H	実習6	
9032_H	47	LD　B, A		
9033_H	0E 03	LD　C, 03H		
9035_H	81	ADD　A, C		
9036_H	32 20 91	LD　(9120H), A		
9039_H	78	LD　A, B		
$903A_H$	91	SUB　C		
$903B_H$	32 21 91	LD　(9121H), A		
$903E_H$	C9	RET		当サブルーチンを終了してMCモニタに戻る.
9500_H	06 10	LD　B, 10H	実習7	
9502_H	78	LD　A, B		
9503_H	3D	DEC　A		
9504_H	32 00 92	LD　(9200H), A		
9507_H	04	INC　B		
9508_H	78	LD　A, B		
9509_H	32 01 92	LD　(9201H), A		
$950C_H$	C3 30 90	JP　9030H		アドレス9030_Hにジャンプ.

このプログラムは，アドレス9000_Hにいきなりジャンプ命令を置き，「実習7」のスタート・アドレス9500_Hへジャンプさせます．「実習7」では最後のリターン命令をジャンプ命令に変更し，「実習6」のスタート・アドレス9030_Hにジャンプさせます．

ではこのプログラムをMCモニタでメモリに書き込み，実行してみましょう．アドレスが連続していない部分がありますので，書き込みには注意してください．



```
-S9000↵ ………Sコマンドでオブジェクト・プログラムを書き込む.
9000 47 C3↵ ┐
9001 F0 00↵ │アドレス9000H に9500H へのジャンプ命令を書き込む.
9002 A1 95↵ ┘
9003 D3 . ↵ ………Sコマンドを終わる.

-D9000 9002↵ ………ダンプして確認.
9000  C3 00 95

-S9030↵      ┐
9030 FF 3E↵  │
9031 A7 15↵  │
 ：（途中省略）│アドレス9030H から実習6のオブジェクト・プログラムを書き込む.
903E 4D C9↵  │
903F 49 . ↵  ┘

-D9030 903E↵ ……ダンプして確認.
9030  3E 15 47 0E 03 81 32 20 91 78 91 32 21 91 C9

-S9500↵        ┐
9500 00 06↵    │
9501 00 10↵    │
 ：（途中省略）  │アドレス9500H から実習7のオブジェクト・プログラムを書き込む.
950C 1A C3↵ *  │(最後の***の3バイトをジャンプ命令に変更してある.)
950D 07 30↵ *  │
950E 80 90↵ *  │
950F FF . ↵    ┘

-D9500 950E↵ ………ダンプして確認.
9500  06 10 78 3D 32 00 92 04 78 32 01 92 C3 30 90

-S9120↵     ┐
9120 E9 00↵ │
9121 CD 00↵ │
9122 98 00↵ │
9123 E9 . ↵ │
            │結果がストアされるメモリを0クリアしておく.
-S9200↵     │
9200 ED 00↵ │
9201 11 00↵ │
9202 40 00↵ │
9203 F0 . ↵ ┘

-G9000↵ ………この実習9のプログラムを実行する. スタート・アドレスは9000H.

-D9120 9122↵ ┐
9120  18 12 00 │
               │実行結果の確認. 当然であるが実習6, 7の場合と同じ.
-D9200 9202↵ │
9200  0F 11 00 ┘

-
```

Figure-8.5.2 実習9(無条件ジャンプ)の実行

実習10 条件ジャンプ(キャリーフラグとゼログラフによる)

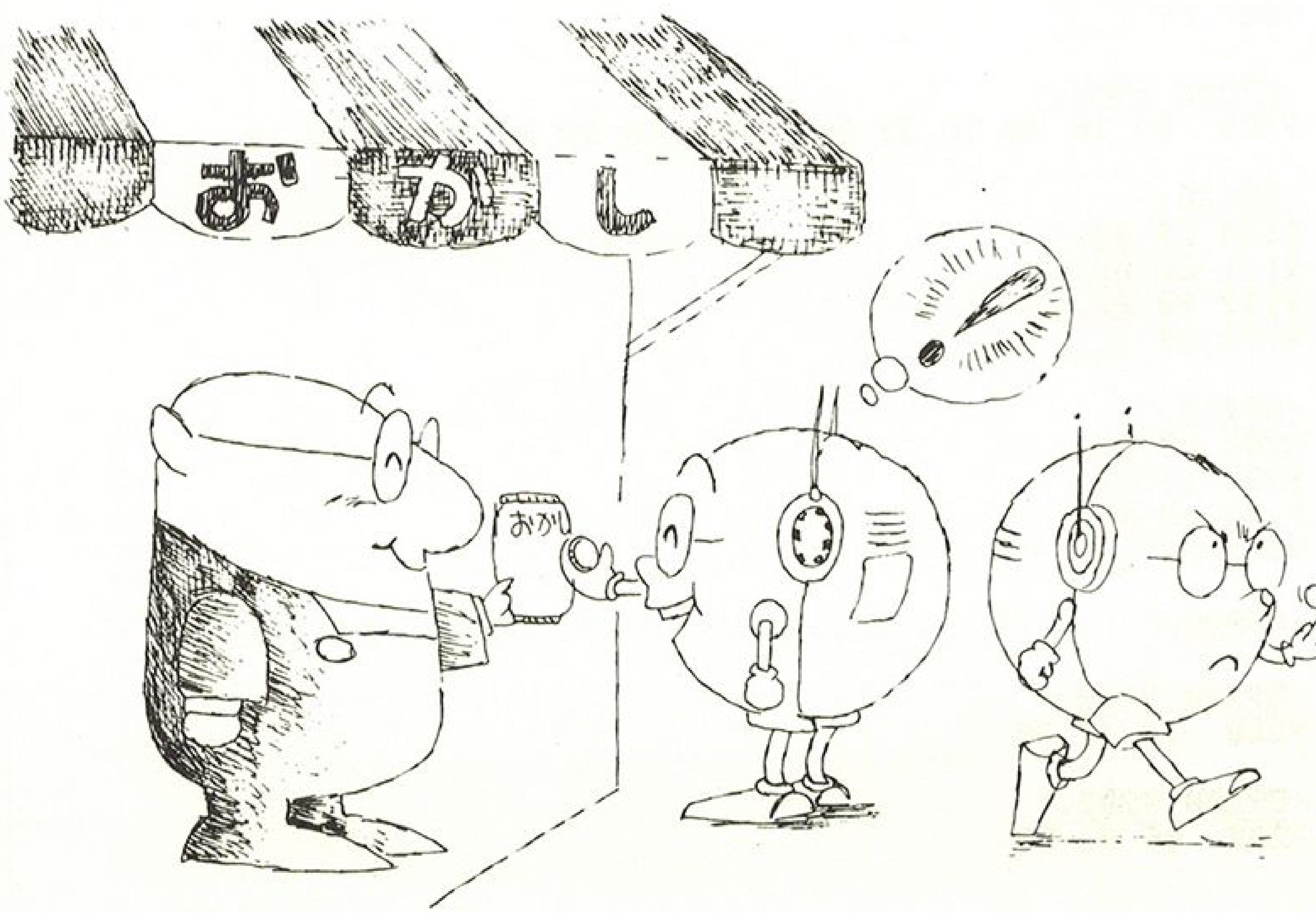
コンペア命令の実習をここで引き継ぎ，条件ジャンプに関係づけて実習解説しましょう．

ここで取り上げる例題としては，前節の「条件分岐について」のFigure-8.4.2に示されている「100円持ってお菓子を買いに行く」問題を考えてみましょう．

このお菓子の問題をちょっと発展させて，次のようなプログラムを作ってみましょう．

100円持ってお菓子を買いに行ったとします．ほしいお菓子がa円であった場合，

- ・100円より高くて買えなかったときは，FF_Hをメモリにストア．
- ・ちょうど100円でおつりがないときは，00_Hをメモリにストア．
- ・100円以下でおつりがあるときは，おつりの額をメモリにストア．



これをコンペア命令と条件ジャンプ命令を使ってプログラミングします．

このプログラムの一例を「実習10その１」として次に示します．しかし，このプログラムは，コンペア命令よりは減算(サブトラクト)命令を使った方がりこうなので，そのあとに「その２」として示します．

コンペア命令を使ったプログラムで，結果をストアするメモリ・アドレスを9100_H,プログラムのスタート・アドレスを9000_Hとして,ハンド・アセンブルした例を次に示します．この例ではお菓子の値段をとりあえず95円と想定して，$5F_H$(＝95)としてあります．

アドレス	マシン語	ラベル	命令	オペランド	説明
			ORG	9000H	
9000_H	3E 64		LD	A,64H	Aレジスタに100(＝64_H)をロード．
9002_H	FE 5F*		CP	5FH*	お菓子の値段(●●)と比較．
9004_H	DA 10 90		JP	C, 買えないアドレス	キャリーフラグが「１」であったら 買えないアドレス にジャンプ．そうでなければすぐ下のステップへ．
9007_H	CA 16 90		JP	Z, ちょうどアドレス	ゼロフラグが「１」であれば ちょうどアドレス にジャンプ．そうでなければすぐ下のステップへ．
$900A_H$	D6 5F*		SUB	5FH*	このステップはおつりがある場合である．Aレジスタの「100」から「●●」を引く．
$900C_H$	32 00 91		LD	(9100H), A	その結果をアドレス9100_Hにストア．
$900F_H$	C9		RET		当サブルーチンを終了してMCモニタに戻る．
9010_H	3E FF	買えないアドレス	LD	A, 0FFH	買えない場合ここに来る．AレジスタにFF_Hをロード．
9012_H	32 00 91		LD	(9100H), A	それをアドレス9100_Hのメモリにストア．
9015_H	C9		RET		MCモニタに戻る．
9016_H	3E 00	ちょうどアドレス	LD	A, 00H	ちょうど100円だった場合ここに来る．Aレジスタに00_Hをロード．
9018_H	32 00 91		LD	(9100H), A	それをアドレス9100_Hのメモリにストア．
$901B_H$	C9		RET		MCモニタに戻る．

合計28バイトのオブジェクト・プログラムになりました．ではこれをMC

＊お菓子の値段を設定している．アドレス9003_Hと$900B_H$の両方の値を変更することにより、お菓子の値段を変えることができる．

モニタでメモリに書き込み，実行してみましょう．

```
-S9000
9000 47 3E
9001 F0 64
9002 A1 FE
9003 D3 5F
9004 10 DA
  ⋮ (途中省略)                      } Sコマンドでオブジェクト・プログラムを書き込む.
9016 4E 3E
9017 3A 00
9018 41 32
9019 43 00
901A 54 91
901B 49 C9
901C 56 .    ………… Sコマンドを終わる.
-D9000 901B ……… ダンプして確認.
9000  3E 64 FE 5F DA 10 90 CA 16 90 D6 5F 32 00 91 C9
9010  3E FF 32 00 91 C9 3E 00 32 00 91 C9
-S9100 ……… 結果がストアされるメモリを0クリアしておく.
9100 E6 00
9101 01 00
9102 28 00
9103 FA .
-G9000 ……… 実習10その1のプログラムを実行する.
-D9100 9102 ………… 実行結果の確認.
9100  05 00 00
      ↑――― 64H－5FH（100円－95円）のおつりの値.
-S9003
9003 5F 64  } お菓子の値段をちょうど100円（64H）に変更してみる.*
9004 DA .
-G9000 ……… 再びプログラムを実行.
-D9100 9102 …. 実行結果の確認.
9100  00 00 00
      ↑――― ちょうど100円なのでおつりなしの「00」.
-S9003
9003 64 65  } お菓子の値段を101円（65H）に変更してみる.*
9004 DA .
-G9000 ……… 再びプログラムを実行.
-D9100 9102 …… 実行結果の確認.
9100  FF 00 00
      ↑――― 買えない場合の「FF」.
-
```

Figure-8.5.3 実習10 その1（条件ジャンプ）の実行例

この問題であれば，コンペア命令を使わないでもう少しよい方法がありますので「実習10その2」として示しておきましょう．各種のフラグは，コンペア命令を実行したときだけでなく，他の演算を行なったときもセットされますので，いろいろな使い方が考えられるひとつの例です．ここではコンペア命令の代わりに減算命令を利用しています．減算命令の後にフラグが変化することをうまく使っています．

CP／M上のアセンブラによりアセンブルした結果のリストで示します．

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

ロード・ オブジェクト・ ラインNo.          ソース・プログラム
アドレス プログラム
9000     3E64          0001               ORG    9000H
                       0002 ;
9000     3E64          0003               LD     A,100   …Hがないので10進数.
9002     D65F          0004               SUB    95
9004     DA0E90        0005               JP     C,KAENAI
9007     CA1490        0006               JP     Z,CHOUDO
900A     320091        0007               LD     (9100H),A
900D     C9            0008               RET
                       0009 ;
900E     3EFF          0010 KAENAI:       LD     A,0FFH
9010     320091        0011               LD     (9100H),A
9013     C9            0012               RET
                       0013 ;
9014     3E00          0014 CHOUDO:       LD     A,00
9016     320091        0015               LD     (9100H),A
9019     C9            0016               RET

Errors                 0
```

Figure-8.5.4 実習10 その2(条件ジャンプ)のプログラム・リスト

「その２」のプログラムでは26バイトのオブジェクト・プログラムになりました．この実行は，「その１」と同様に行なえますので各自で試みてください．

＊お菓子の値段を100円以下の別の値に変更する場合は，アドレス9003Hとともに900BHも同じ値に変更する必要がある．ちょうど100円にする場合や，おつりのない値に変更する場合はアドレス900BHを通らないので，ここでは省略している．

相対(リラティブ)アドレス・ジャンプ

Z-80には相対アドレス・ジャンプの命令があります．これに対し，実習9，10で使用した「JP」命令は「絶対アドレス・ジャンプ」と呼んでいます．絶対アドレス・ジャンプ命令は，「アドレスXXXX$_H$へジャンプせよ」というようにジャンプ先アドレスが絶対的に定まっていますが，相対アドレス・ジャンプ命令は「ここよりXX番地先または後にジャンプせよ」というようにジャンプ先アドレスを，自分自身の命令があるアドレスを基準に相対的に表わす方法をとります．

この命令は特にハンド・アセンブルを行なう場合など非常にめんどうなので，本書では，取り扱いませんが，その概念を一応解説しておきましょう．

具体例として，アドレス9000$_H$にあるジャンプ命令によって，前方のアドレス9008$_H$にジャンプする場合を例に，絶対アドレス・ジャンプ命令と，相対アドレス・ジャンプ命令とを対比して次に示します．

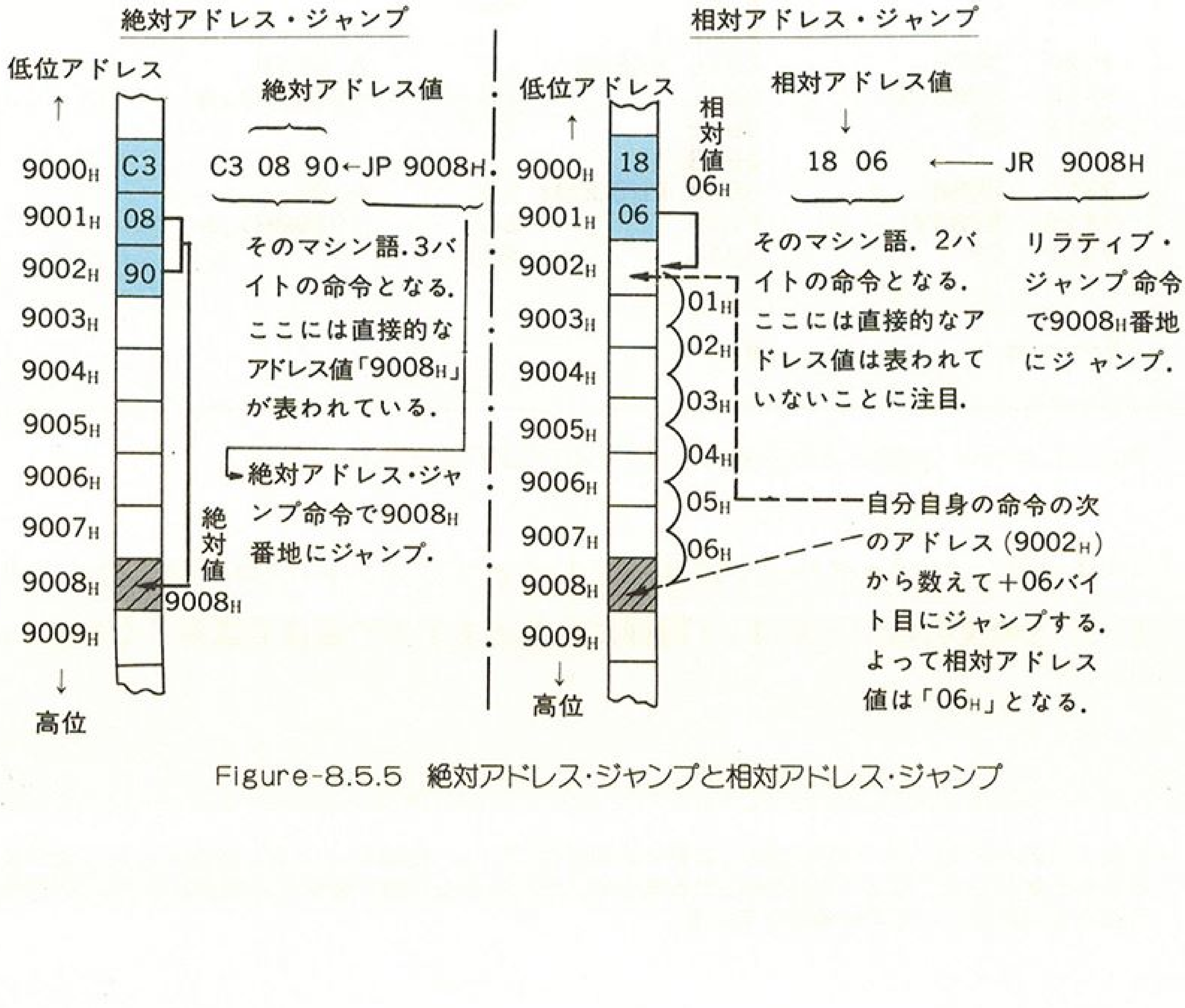


Figure-8.5.5 絶対アドレス・ジャンプと相対アドレス・ジャンプ

この相対ジャンプの例は，アドレスの高位の方にジャンプする例ですが，もちろん低位の方にジャンプすることもできます．この場合の相対アドレス値を計算するのは少々めんどうになりますので本書ではふれませんが，相対アドレスで指定可能な範囲は，JR命令自身の１バイト目のアドレスから高位方向に129バイト，低位方向に126バイトの間です．

相対アドレス・ジャンプ命令を使うと，２バイト命令であるため絶対アドレス・ジャンプ命令より１バイト分メモリの節約にはなります．さらに大切なのは，＋129バイトから－126バイトの範囲の相対アドレス・ジャンプだけを利用することで，プログラムを「リロケータブル」にできることです．このリロケータブル，つまりメモリ上を「移動することが可能」という意味を具体的に説明してみましょう．

さきほどの「実習10その２」のアセンブルリストを見てください．このオブジェクト・プログラムは，メモリ上のアドレス9000_Hから書き込まなければ動作しません．それはラインNo.５～６のジャンプ命令が絶対アドレス・ジャンプであり，それぞれ$900E_H$および9014_Hにジャンプすることが絶対的に決まっているからです．よってたとえ１番地たりともずれてオブジェクト・プログラムが書き込まれると，プログラムは働きません．このオブジェクト・プログラムはアドレス9000_Hからメモリに書き込まなければ働かないのです．

ここでこのふたつのジャンプ命令を，相対アドレス・ジャンプ命令に変えてアセンブルした結果を示します．

このアセンブルリストは，オブジェクト・プログラムのスタート・アドレスを一応9000_Hとしてアセンブルしてありますが，このプログラムのどこにも絶対的なアドレスに関する命令はありません（結果をストアするアドレスを除いて）．ジャンプ命令は相対アドレス・ジャンプに変更されて，絶対的アドレス値は姿を消しています．よってこのオブジェクト・プログラムは，メモリ上のどこへ書き込んでも動作可能です（結果をストアするアドレスに重複しないように）．つまり「リロケータブル」ということです．試みにこの「実習10その３」のオブジェクト・プログラムを，アドレス9500_Hからメモリに書き込ん

で実行してみてください．同様にA000ₕからも試みてください．

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

                        0001          ORG     9000H
                        0002 ;
9000  3E64              0003          LD      A,100
9002  D65F              0004          SUB     95
9004  3806              0005          JR      C,KAENAI
9006  280A              0006          JR      Z,CHOUDO
9008  320091            0007          LD      (9100H),A
900B  C9                0008          RET
                        0009 ;
900C  3EFF              0010 KAENAI:  LD      A,0FFH
900E  320091            0011          LD      (9100H),A
9011  C9                0012          RET
                        0013 ;
9012  3E00              0014 CHOUDO:  LD      A,00
9014  320091            0015          LD      (9100H),A
9017  C9                0016          RET

Errors                  0
```

相対ジャンプ命令

（このオブジェクト・プログラムは，メモリ上のどの位置にロードしても動作可能．ただし結果をストアするアドレス9100ₕに重複しないこと．）

Figure-8.5.6　実習10 その3 相対アドレス・ジャンプ命令を使う

ここで実行速度のことに関して少しふれましょう．「JP」命令は3バイト，一方，「JR」命令は2バイトでメモリの節約にはなりますが，実行スピードは遅くなるのです．「JR」命令の方がメモリの節約になっただけ，実行も速くなると思い込んでいる人が多いのですが，これは誤りですので注意してください．「JR」命令の方が「JP」命令より実行速度が約20%低下します．ただし条件ジャンプの場合は，その条件によっては速い場合もあります．

それに，プログラムが占有するメモリを節約しなければならない特別の理由があれば別ですが，そうでなければ相対（リラティブ）ジャンプ命令を使う必要はありません．ジャンプ命令だけをリラティブ形式にしても，現実的にはあまりメリットはありません．むしろ使わない方があとあとのトラブルも少ないのです．

8.6

●コール，リターン命令

コール命令はサブルーチンをコールするための命令で，分岐命令のひとつです．次にコール命令の動作を図解で示します．

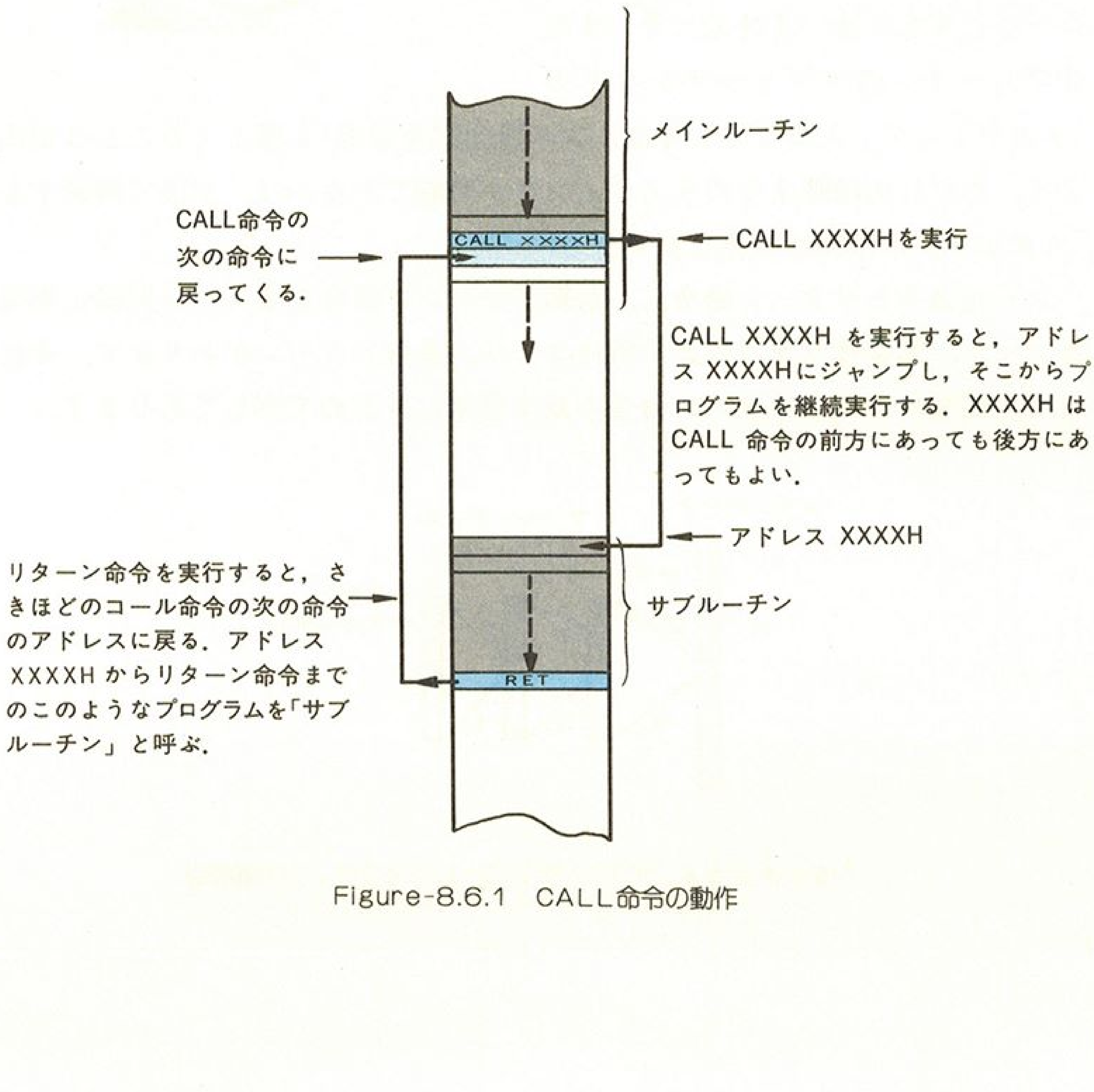


Figure-8.6.1　CALL命令の動作



この図からもわかるように，コール命令は自動的にもとの位置に戻ってくることができるジャンプ命令と考えてもよいでしょう．もとの位置（コール命令の次の命令）に戻る「きっかけ」となるのがリターン命令の実行であるわけです．

コール命令は，図のようにメインルーチンからサブルーチンをCALLしますが，サブルーチンの中にまたコール命令があって，さらにサブルーチンをコールしてもかまいません．そのまたサブルーチンのサブルーチンなどと「ネスティング」（入れ子とも呼ぶ．次の概念図を参照）を深くすることは自由です．ただし何段階までのネスティングが可能であるかは，次節で解説する「スタック」の余裕に関係します．

コール命令とリターン命令は，前節のジャンプ命令とまったく同様に無条件コール，無条件リターンと，条件コール，条件リターンがあります．それらの主要なコール，リターン命令の表は巻末にまとめて示してあります．

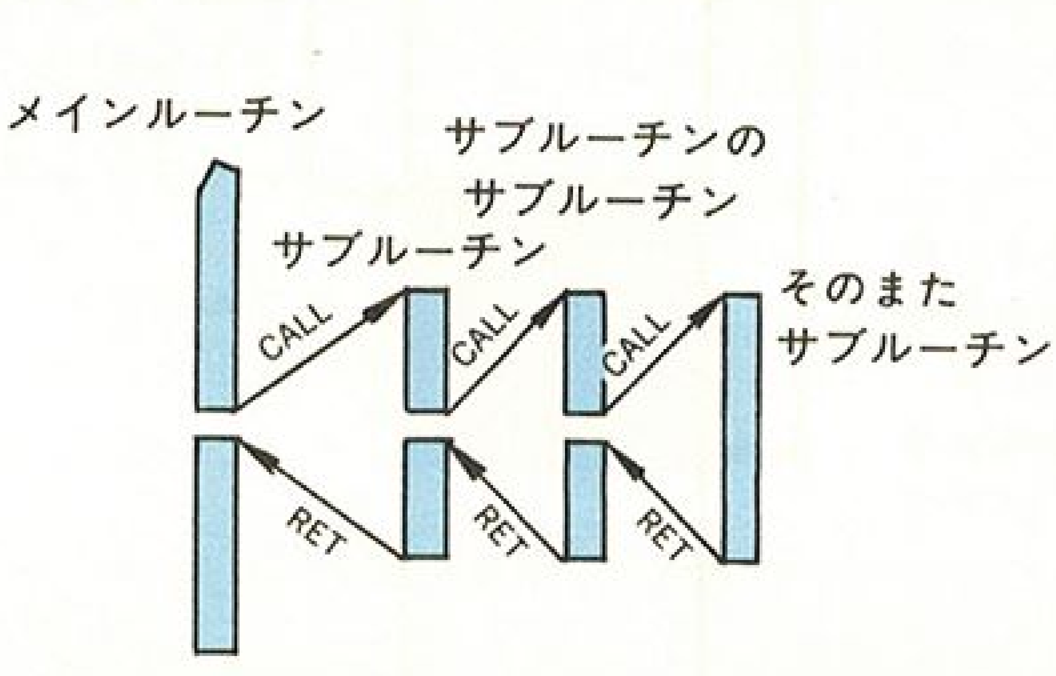


Figure-8.6.2　サブルーチンコールのネスティング概念図

無条件コール命令は,

CALL　○○●●…………マシン語では「CD　●●　○○」の3バイトの命令.

条件コール命令は，条件ジャンプ命令とまったく同様に考えて，各フラグ条件を×とすると,

CALL　×，○○●●……マシン語では×＝Z（つまりZ(ゼロ)フラグが「1」でコール）の場合「CC　●●　○○」の3バイトの命令, ×＝C（つまりCY(キャリー)フラグが「1」でコール）の場合「DC　●●　○○」の3バイトの命令．その他は表を参照.

無条件リターン命令は,

RET………………マシン語では「C9」の1バイトの命令.

条件リターン命令は，条件コール命令や条件ジャンプ命令とまったく同様に考えて，各フラグ条件を×とすると,

RET　×…………マシン語では×＝Z（つまりZ(ゼロ)フラグが「1」でリターン）の場合　「C8」の1バイトの命令, ×＝C（つまりCY(キャリー)フラグが「1」でリターン）の場合「D8」の1バイトの命令.

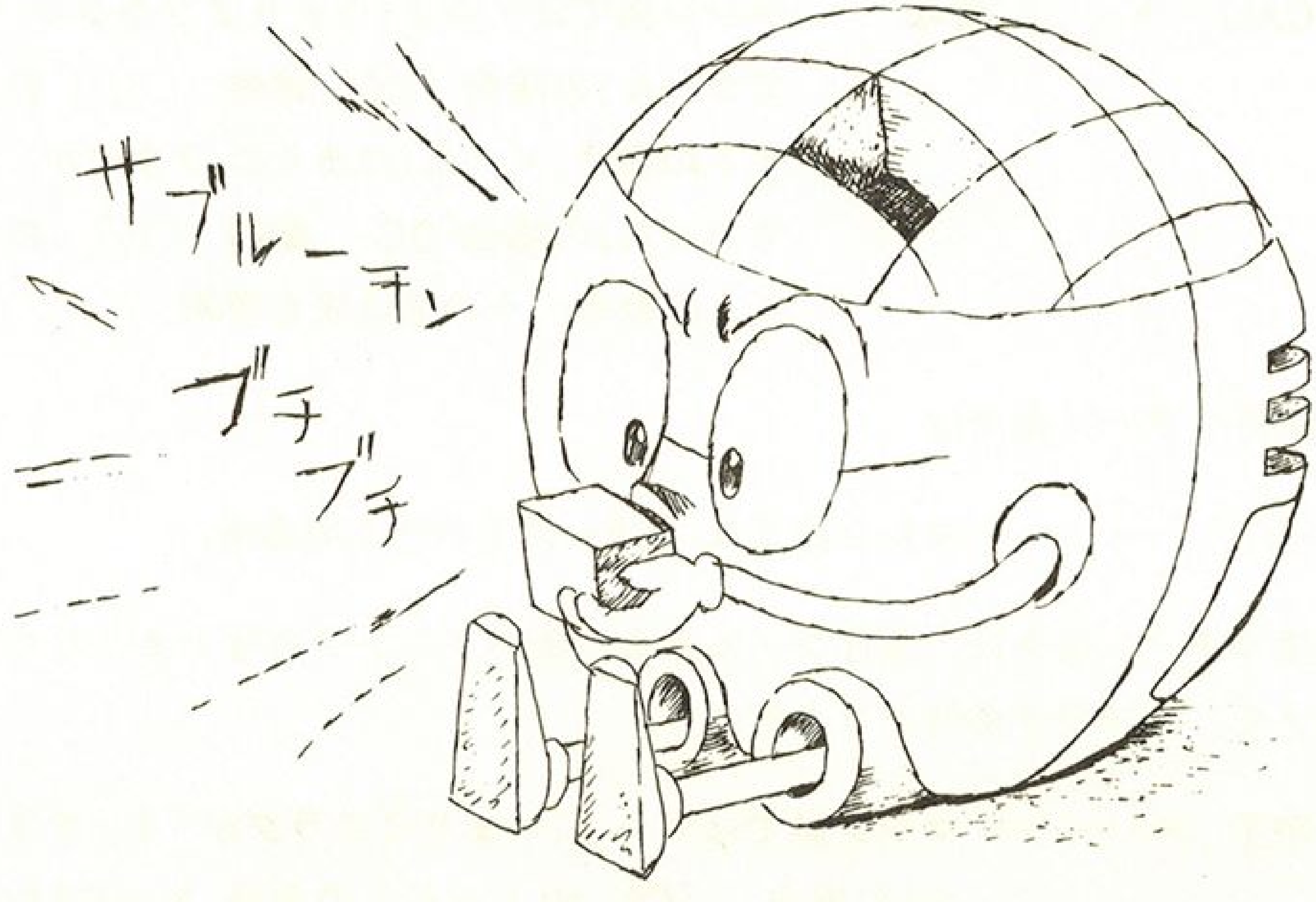
という形式で表わします.

コール命令は，プログラミング技法上たいへん重要なもので，良いプログラムを書くということは，このサブルーチンコールをうまく使い，いかに整然とわかりやすいプログラムの流れとするかがそのポイントになります*.

またサブルーチンとは，プログラム上のいろいろなところで同じ作業(動作)が必要な場合など, その作業をリターン命令で終わるひとつのプログラム

*「構造化プログラミング」というプログラミング技法の基本でよくいわれていることのひとつ．極力適当な機能に分けたサブルーチン化を計り，メインルーチン，サブルーチン，そのまたサブルーチンというように構造的にプログラムを組上げていくと良い.

として，適当なメモリ・アドレスに配置したもののことです．メインルーチンでその作業が必要になったときにコール命令で呼び出して，その作業をさせるというような使い方をします．つまりその作業が何回あろうと，その作業のプログラムは，メモリ上の任意の位置にただひとつ用意しておけばよいわけです．



たとえばスクリーンに文字を表示する場合など,「Aレジスタに持っている文字コードをスクリーンのカーソルの位置に表示する」というひとつの作業プログラムをサブルーチンにしておくと，文字列をスクリーンに表示する必要があった場合は，その文字列の頭から順番に1文字ずつ文字列のデータをAレジスタにロードしてはその「1文字スクリーン表示サブルーチン」をコールすることをくり返せばいいわけです．9章でこの1文字表示サブルーチンを使った例題を示して解説していますので，のちほどでも参照してください．

実習11　サブルーチンとサブルーチンコール

「実習10その２」のお菓子の値段の問題を使って，コール命令とリターン命令の実習を行ないます．実習10の内容を忘れてしまった方はもう一度参照してください．

当実習では次のような判別作業をひとつのサブルーチンとしてプログラムします．

> お菓子屋さんへ行って，ほしいお菓子の値段(Bレジスタの内容)と，手持ちの100円とを比較して，買えない場合はFF$_H$，ちょうど100円でおつりがない場合は00$_H$，おつりがある場合はおつりの値をＡレジスタにセットする．

メインルーチンでは，Bレジスタにいろいろなお菓子の値段をセットして，この判別のサブルーチンをコールします．コールから戻ると，Ａレジスタは判別の結果がはいっていますので，それを適当なメモリにストアします．これを数回くり返すようなプログラムを作ってみましょう．

まず最初に上記の判別作業をするサブルーチンを作ってみましょう．

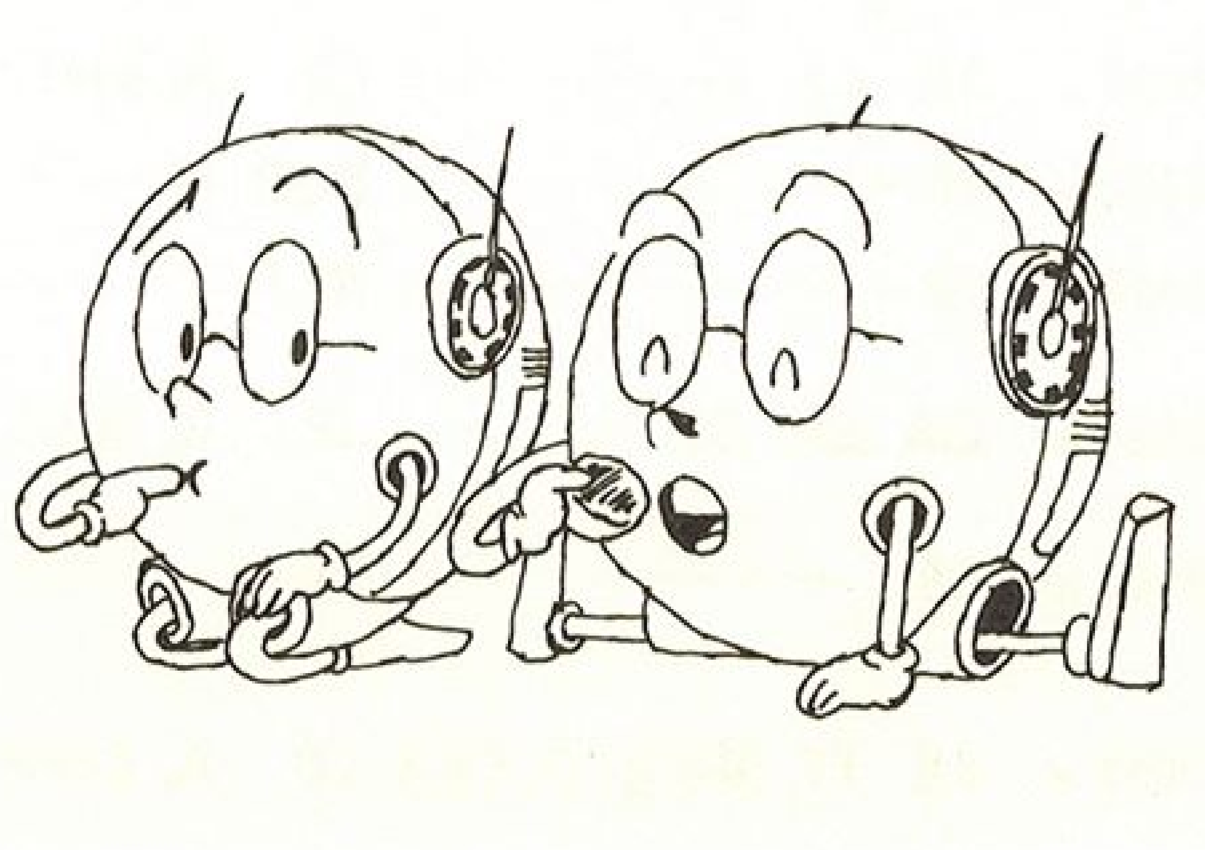
Ｂレジスタにお菓子のいろいろな値段をロードしてコールすると，上記の枠で囲んだような働きをするサブルーチンです．

```
                                ORG  9050H
9050H  3E 64                    LD   A, 64H ……… 手持ち額100円(100=64H)
                                                をＡレジスタにロード．
9052H  90                       SUB  B ………… Ａレジスタ－Ｂレジスタ．
                                                結果はＡレジスタに残る．
9053H  C8                       RET  Z ………… Z(ゼロ)フラグが「1」(結
                                                果が00H)の場合リターン．
                                                Ａレジスタ＝00H．
9054H  DA 58 90                 JP   C,[買えないアドレス] …… C(キャリー)フラグが「1」
                                                (結果がマイナス)の場合
                                                [買えないアドレス]にジャンプ．
9057H  C9                       RET ……………… 上のいずれでもない場合
                                                (おつりがある場合)リター
                                                ン・おつりの額はＡレジス
                                                タに残っている．
9058H  3E FF  [買えないアドレス] LD   A, 0FFH …… 買えない場合ここにジャン
                                                プして来る．Ａレジスタに
                                                FFHをロード．
905AH  C9                       RET ……………… リターン．Ａレジスタ＝
                                                FFH．
```

この判別サブルーチンを，アドレス9050_Hのメモリに置き，9000_Hからのメインルーチンで次のようにコールしましょう．

```
                                    ORG   9000H
9000H  06 C8     <------------  LD    B,0C8H ········· 200円(200＝C8H）をBレジスタにロード．
9002H  CD 50 90  <------------  CALL  9050H  ········· 200円の場合はどうかな？判別サブルーチンをコール．
9005H  32 00 91  <------------  LD    (9100H),A ······ 結果をアドレス9100Hのメモリにストア．
9008H  06 64     <------------  LD    B,64H ·········· 100円(100＝64H）をBレジスタにロード．
900AH  CD 50 90  <------------  CALL  9050H  ········· 100円の場合はどうかな？判別サブルーチンをコール．
900DH  32 01 91  <------------  LD    (9101H),A ······ 結果をアドレス9101Hのメモリにストア．
9010H  06 32     <------------  LD    B,32H ·········· 50円(50＝32H）をBレジスタにロード．
9012H  CD 50 90  <------------  CALL  9050H  ········· 50円の場合はどうかな？判別サブルーチンをコール．
9015H  32 02 91  <------------  LD    (9102H),A ······ 結果を9102Hのメモリにストア．
9018H  C9        <------------  RET          ········· 当メインルーチン終了．MCモニタに戻る．
```

このようなサブルーチンとメインルーチンができました．これらのオブジェクト・プログラムをMCモニタでメモリに書き込み，実行して結果を確認してみましょう．



```
-S9050 ↵
9050 73 3E ↵  ┐
9051 19 64 ↵  │
9052 E9 90 ↵  │
9053 31 C8 ↵  │
9054 00 DA ↵  │
9055 F0 58 ↵  ├ Sコマンドでサブルーチンのオブジェクト・プログラムを書き込む.
9056 F5 90 ↵  │
9057 E5 C9 ↵  │
9058 D5 3E ↵  │
9059 C5 FF ↵  │
905A CD C9 ↵  ┘
905B 6C . ↵ ……Sコマンドを終わる.

-D9050 905A↵ ……ダンプして確認.
9050  3E 64 90 C8 DA 58 90 C9 3E FF C9

-S9000↵
9000 47 06 ↵  ┐
9001 F0 C8 ↵  │
9002 A1 CD ↵  │
9003 D3 50 ↵  │
9004 10 90 ↵  │
9005 F3 32 ↵  │
9006 3A 00 ↵  │
9007 01 91 ↵  │
  ：(途中省略)  ├ Sコマンドでメインルーチンのオブジェクト・プログラムを書き込む.
9012 C9 CD ↵  │
9013 5B 50 ↵  │
9014 50 90 ↵  │
9015 52 32 ↵  │
9016 4E 02 ↵  │
9017 3A 91 ↵  │
9018 41 C9 ↵  ┘
9019 43 . ↵ ……Sコマンドを終わる.

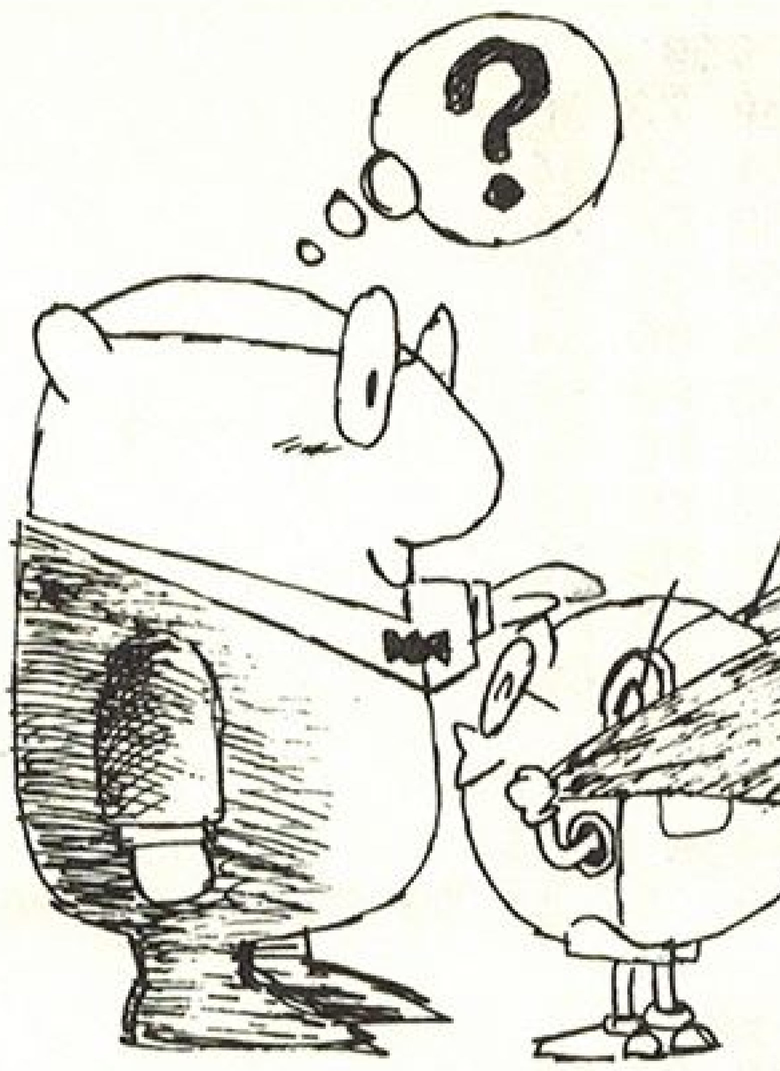
-D9000 9018↵ ……ダンプして確認.
9000  06 C8 CD 50 90 32 00 91 06 64 CD 50 90 32 01 91
9010  06 32 CD 50 90 32 02 91 C9

-S9100↵ ……結果がストアされるメモリを0クリアしておく.
9100 E6 00↵
9101 01 00↵
9102 28 00↵
9103 FA 00↵
9104 3A . ↵

-G9000↵ ……実習11のプログラムを実行する. スタート・アドレスは9000H.

-D9100 9103↵ ……実行結果の確認.
9100  FF 00 32 00
      ↑  ↑  ↑
-  買えない  ちょうど  64H－32H＝32H(100－50＝50)
```

Figure-8.6.3　実習11(サブルーチンとサブルーチンコール)の実行



「実習11」は，無条件コール，無条件リターンと，Zフラグによる条件リターンの命令が含まれていますが，条件コールは含まれていません．しかしこれは「実習10」の条件ジャンプと同様な考え方で，簡単に利用できますので，各自でいろいろと試みてください．

今回の実習で使用した判別サブルーチンは，おつりがある場合とおつりが0円の場合を同じことと考えると，もっとスマートにまとめることができます．参考までにそのプログラムをFigure-8.6.4に示しておきますので，比較してください．もちろんこの新しいサブルーチンを使っても「実習11」のプログラムは同様に働きますので試みてください．

コール命令でサブルーチンをコールした場合，サブルーチン内のリターン命令でなぜもとのメインルーチンに戻れるのか．これは「スタック」という重要な概念と関係していますので，次にこの概念を解説します．

```
                    0001        ORG     9050H
                    0002 ;
9050    3E64        0003        LD      A,100
9052    90          0004        SUB     B
9053    D0          0005        RET     NC
                    0006 ;
9054    3EFF        0007        LD      A,0FFH
9056    C9          0008        RET
```

Aレジスタに100(64H)をロード．

Aレジスタの内容からBレジスタの内容を引く．

Cy(キャリー)が「1」でない場合(Bレジスタの値が100以下の場合)当サブルーチンを終わりメインルーチンに戻る．

このステップに来る場合は，Cyが「1」，つまりBレジスタの値が101以上のとき．よってAレジスタにFFHをセットしてメインルーチンに戻る．

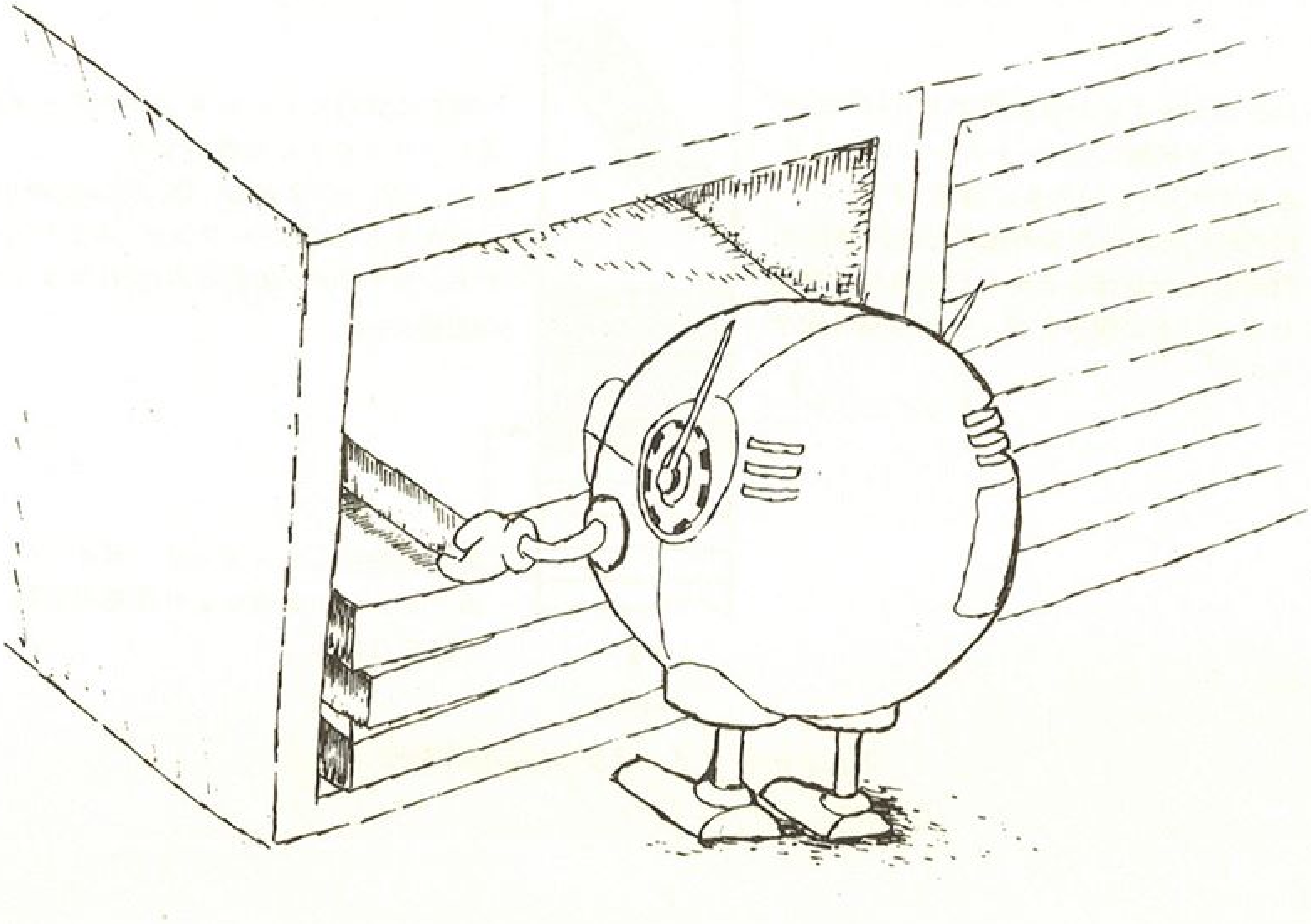
Figure-8.6.4　買える/買えない/おつりない/判別サブルーチン(その2)

8.7

●スタックの概念とスタック・ポインタ

またひとつ，コンピュータのアーキテクチャ上の重要な話をしなければなりません．「スタック」と呼ばれる概念です．

プログラムを組んでいくと，あるいくつかのレジスタの内容を一時的に保存したい場合がよくあります．つまり，「あるレジスタに今後使用するデータがはいっているけれど，今そのレジスタを別の目的で使用しなければならない．そのレジスタを使用する間だけもとのデータをどこかに保存し，レジスタの使用が終われば直ちにもとのデータに戻したい」という場合です．



これはサブルーチンコールの場合などによくある問題です．

このような各レジスタの内容を一時的にメモリへ保存(退避)することとその復帰(保存されているメモリからもとのレジスタへのロード)を処理する機能としてスタックの概念があり，それを処理する便利な命令として「PUSH(プッシュ)」と「POP(ポップ)」の命令が用意されているのです．

もちろんロード命令を使って，適当なアドレスのメモリにレジスタの内容をストアして，それを再びもとのレジスタにロードしてもよいのですが，「PUSH」「POP」の命令はそれらを自動的に行なってくれるもので，しかも1バイトの命令なのです．

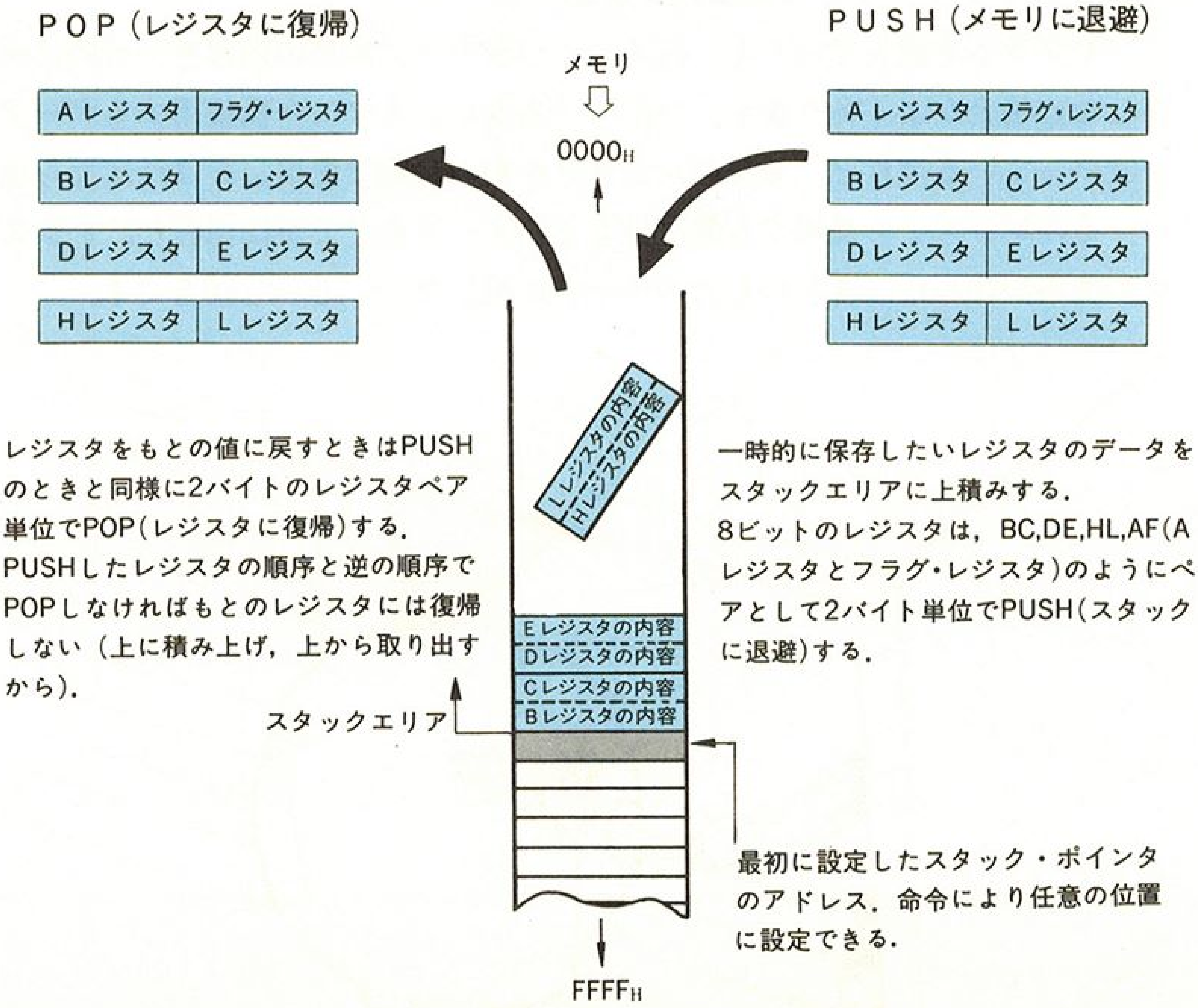


Figure-8.7.1 スタックの基本動作

スタックとは物理的には，ユーザー側が任意に指定するメモリ上の特定のエリアです．このエリアに一時的に保存したいレジスタのデータを上積みしていき，必要なときにそれを上から順に取り出し，レジスタに戻します．その動作の様子をFigure-8.7.1に示しています．

次にスタックに関するいくつかの命令を実行した場合，スタックはどのような動きをするかを説明してみましょう．

LD　SP ,XXXXH …スタック・ポインタ(SP：16ビットのレジスタ)に2バイトの値$XXXX_H$をロード(セット)せよ．(SPの設定)

PUSH　rr ………………レジスタペアrr(たとえばBC，HLなどの16ビットとしての組合せ)の内容をスタックに積む．(退避)

POP　rr ………………スタックの上から2バイトを取り出し，レジスタペアrrにロードする．(復帰)

これらの命令は，のちほど実習解説を行なう「16ビット・ロード命令」と呼ばれるグループのものです．これまでに登場した「8ビット・ロード命令」が，レジスタ↔メモリ間などのデータのやりとりを8ビット(1バイト)単位で行なったのに対し，それを16ビット(2バイト)単位で行なうものです．ここでFigure-7.2.1の「Z-80CPUの内部レジスタ」を参照してください．今までは，B，C，D，E，H，Lレジスタなどを8ビットの単独のレジスタとして扱ってきましたが，16ビット・ロード命令ではこれを，BC，DE，HL，のようにペアにして，16ビットのひとつのレジスタとして扱います．Aレジスタは，フラグレジスタとペアにして，AFレジスタとして16ビットで扱います．そのほかのことについては8.9章の「16ビット・ロード命令」を参照してください．



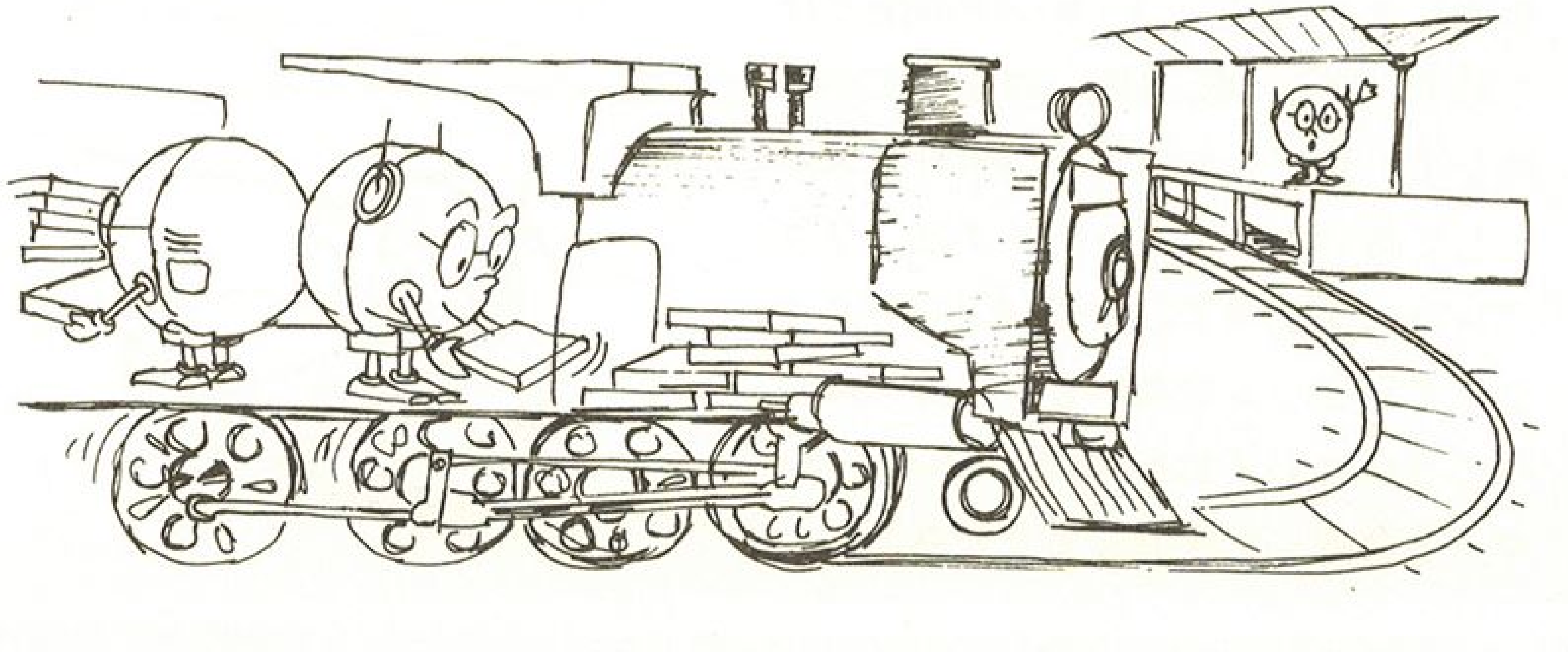
ここでスタック・ポインタ(SP)について説明をしておきましょう．SPは，Figure-7.2.1の「Z-80CPUの内部レジスタ」に示されているように，16ビットの専用レジスタのひとつです．このSPには，PUSH命令やPOP命令それにコール，リターン命令を実行する場合，メモリ上のどのアドレスを使ってデータのストアやロードをすればよいか，そのアドレスを指し示すために，その時点その時点でのアドレスデータがセットされています．

最初のSPの設定は，「LD　SP ,XXXXH」で行ないますが，その後はPUSH命令やCALL命令が実行されれば自動的にSPの内容は－2され(つまり2バイト分上積みされるから)，POP命令やRET命令が実行されれば+2されます(つまり2バイト分取り出されるから)．

次の図は，「DEとHLレジスタペアの内容をスタックエリアに退避しておき，サブルーチンをコールして，サブルーチンの中でDEやHLのレジスタを使う．その後，リターン命令でサブルーチンからメインルーチンに戻ったあと，退避していたDEとHLレジスタペアの内容をもとに復帰させる」という一連の操作を図解したものです．

この図からわかるように，スタックはPUSH，POP命令だけではなく，CALL命令の実行の際にもRET命令によりメインルーチンに戻るアドレスを一時的に記憶する場所としてCPUの内部で自動的に使われているのです．

「実習15」でもPUSH，POPを使った簡単なプログラムを実習していますので参照してください．



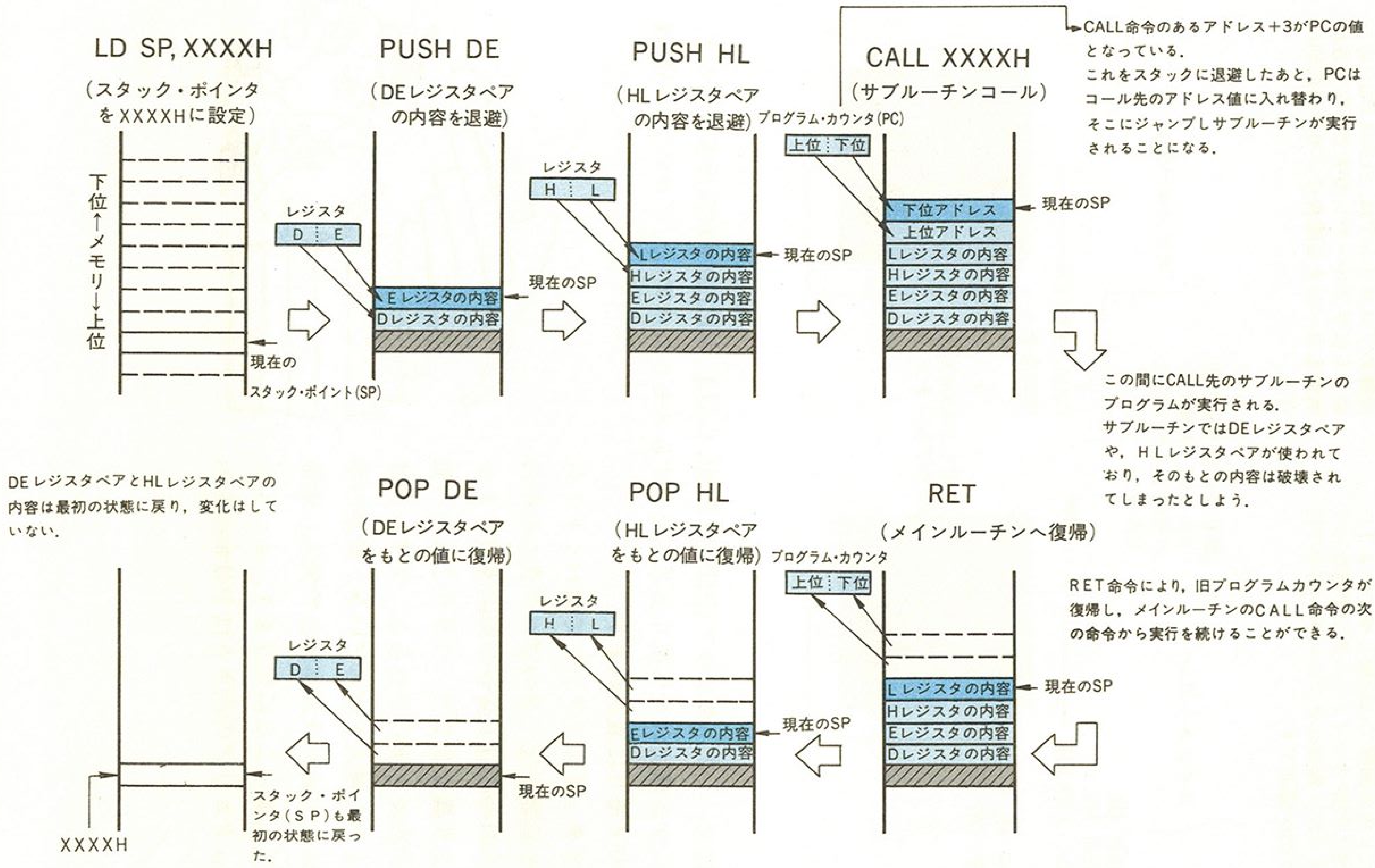
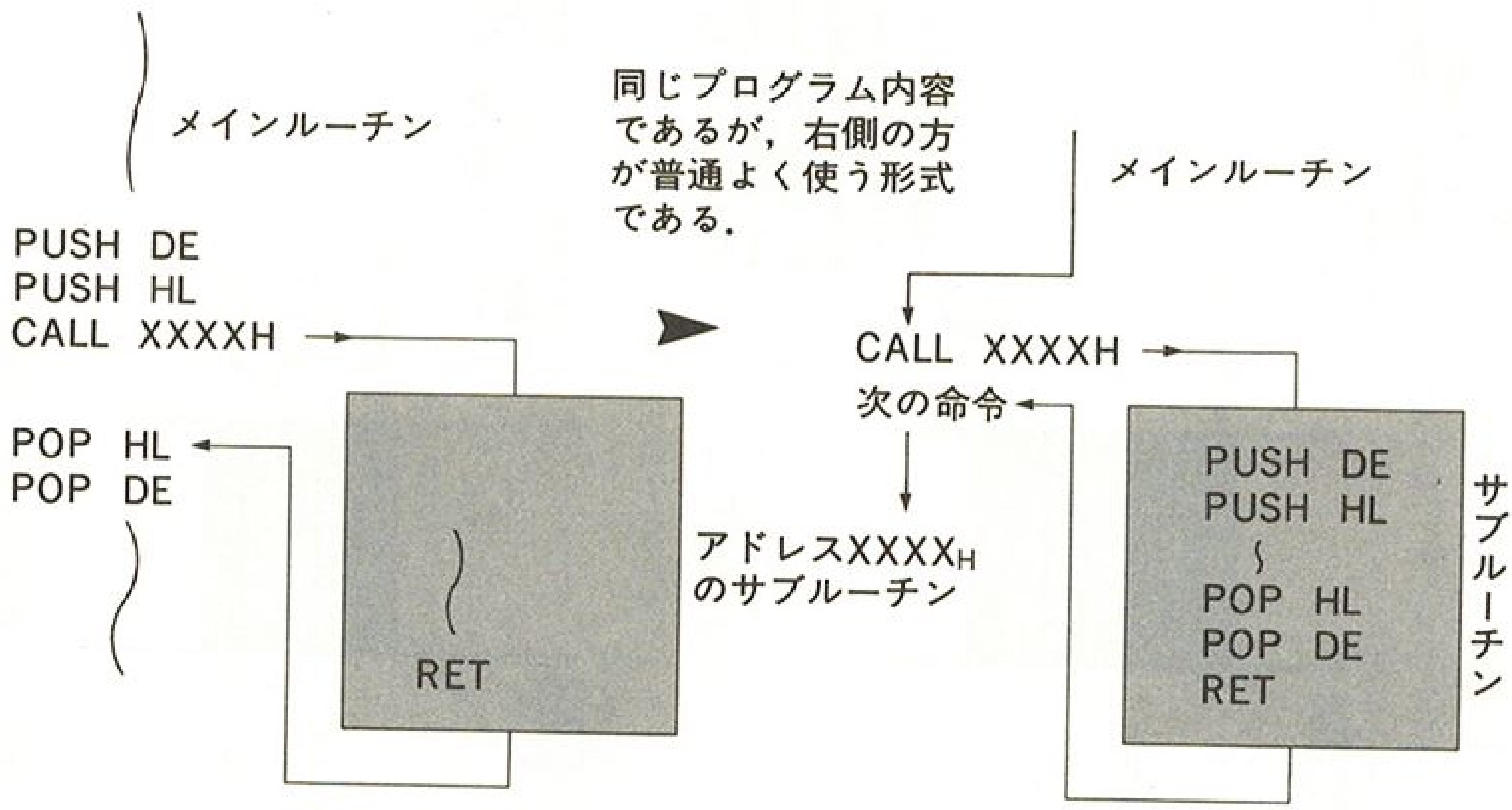


Figure-8.7.2 スタックに関する各種命令を実行した場合のスタックの動き

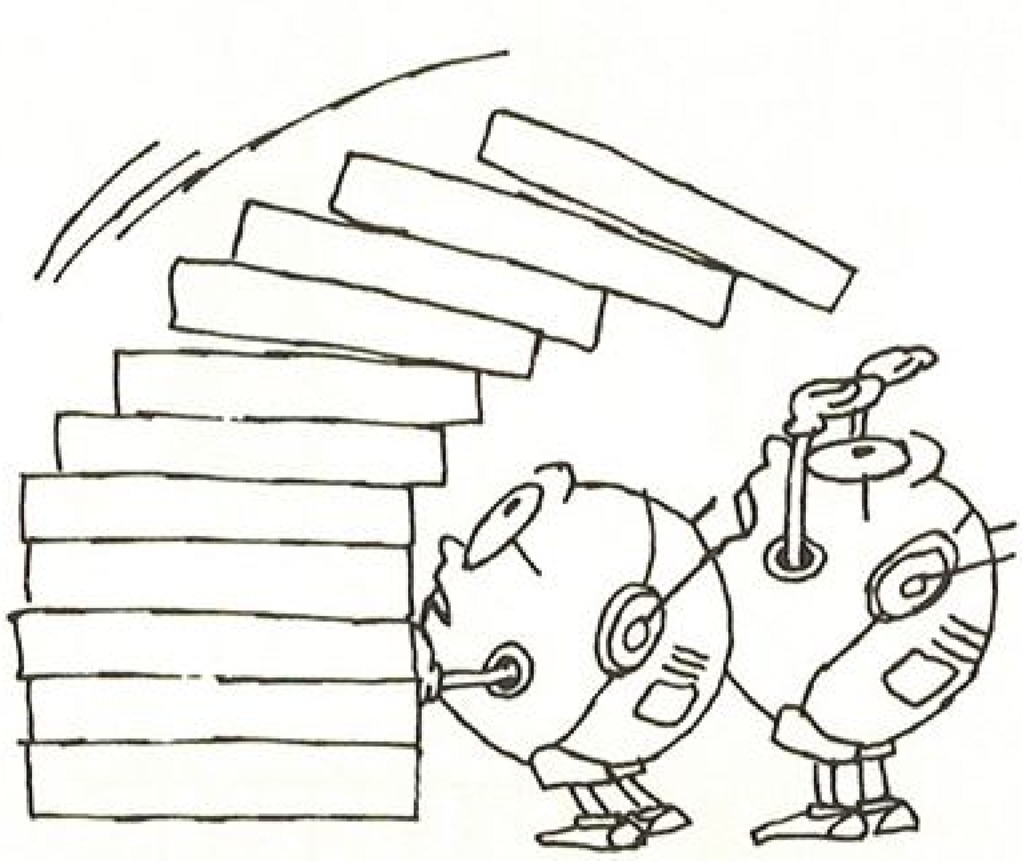
ただし先の図解では理解しやすいように PUSH, POP をメインルーチンで行なっていますが，実際はサブルーチンの中で行なうのが普通です．

実際のプログラムでは，PUSH, POP をこのようにサブルーチンの中で行なう方が何度もコールする場合にメモリの節約になる．



通常，スタックエリア（PUSH, POP, CALL, RET命令が実行されたとき，データが積まれたり取り出されたりするメモリエリア）は，プログラムの最後部のメモリに置かれます．

スタックエリアとして何バイトぐらいが必要かは，そのプログラムの中のPUSHやCALL命令が何回連続するかにより異なります．つまりこれらの命令が次々と実行されると，スタックはどんどん積み上がっていき，最悪の場合は，プログラム本体まで侵入し破壊してしまうことになります．よってスタックエリアの大きさは十分余裕をみ



て，普通のプログラムであれば32バイト（PUSHやCALL命令が16回連続してもOK）とか，64バイト（同じく32回までOK）ぐらい確保しておけばよいでしょう．

スタックエリアのメモリ上の配置の様子を次に示します．

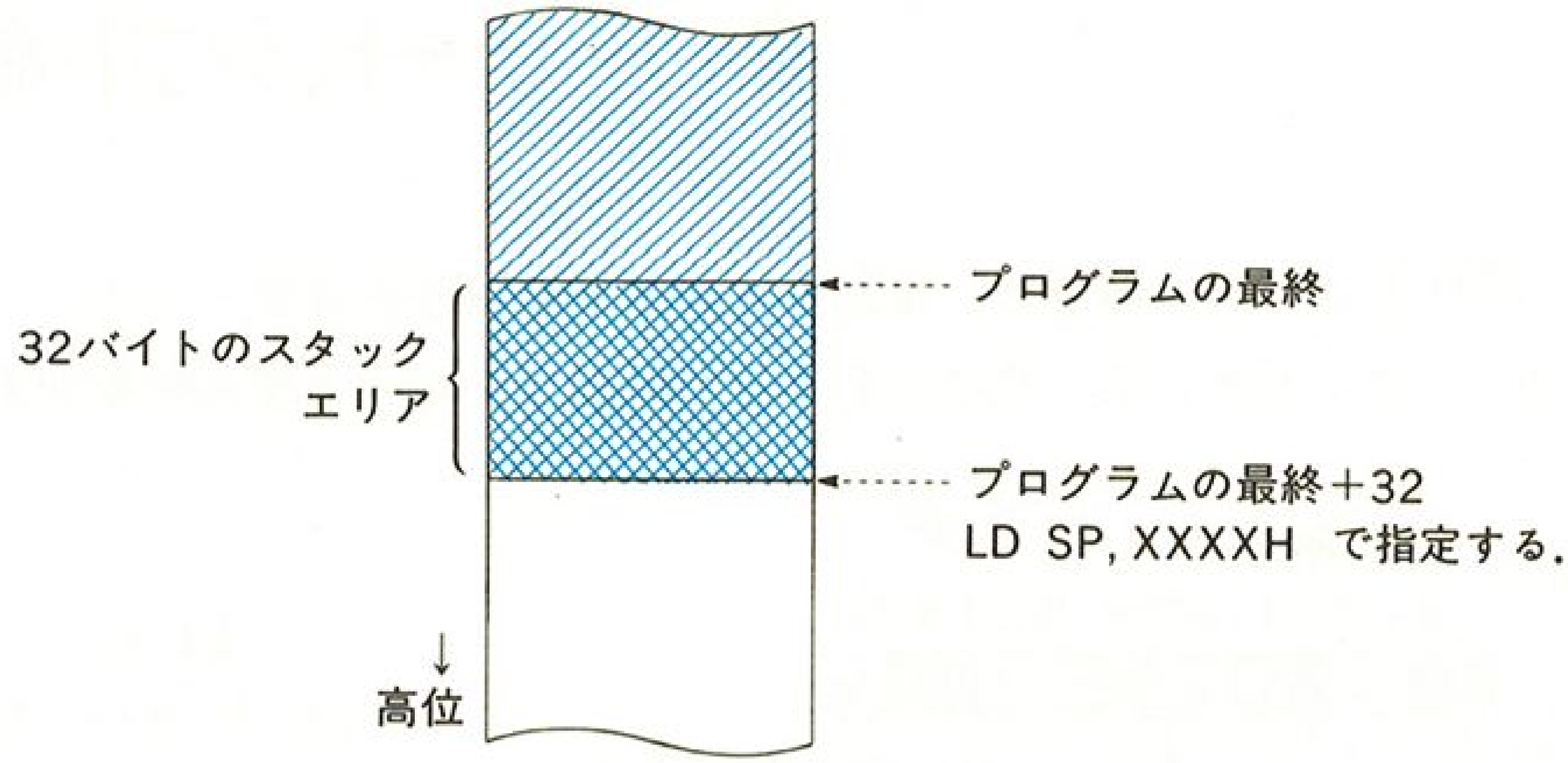


Figure-8.7.3 スタックエリアの配置

マシン語のプログラムを作成する場合は，普通，そのプログラムの冒頭で独自のスタック・ポインタを設定します（LD SP, XXXXH命令で行なう）．しかし，本書の実習のように，モニタ上からそのプログラムを実行して，再びモニタに戻る場合は，単純にLD SP, XXXXHを実行したのではモニタに戻れなくなってしまいます．正しく動作させるためには，BASICの内部で使用している旧スタック・ポインタの退避と復帰のちょっとしたプログラムを書かなければなりません．

よって本書の実習プログラムでは，独自のスタック・ポインタを設定していません．MCモニタのGコマンドは，BASIC上のマシン語コール文によって実行されますので，スタックに関するマシン語プログラムを実行した場合のスタック・ポインタは，BASICが使っているスタック・ポインタを継続して使用することになります．

8.8

●ローテート，シフト命令

ローテートとシフト命令は，任意のレジスタの内容を8ビットの「ビットパターン」として考えて，それを右あるいは左に1ビットずらす命令です．

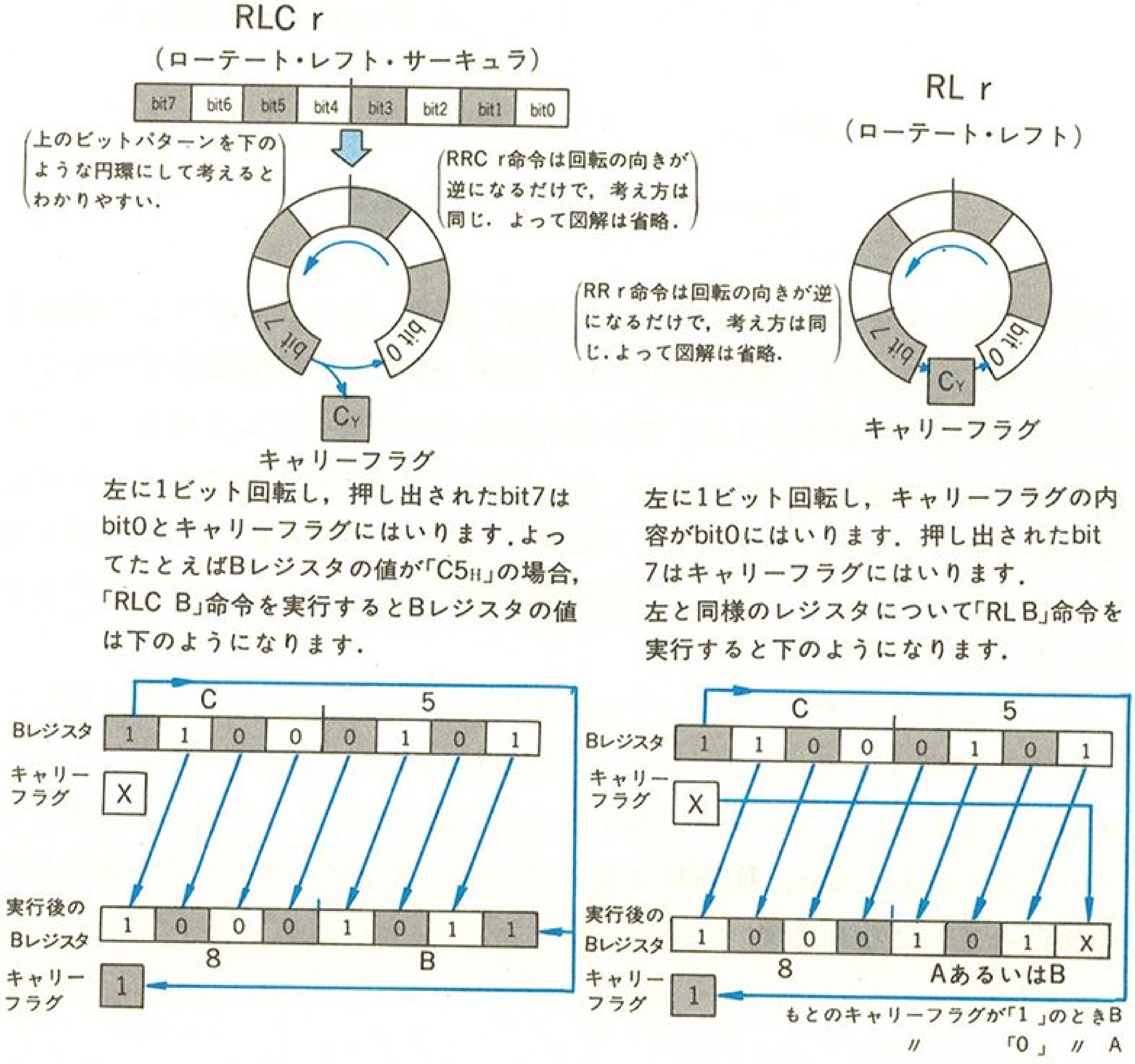


Figure-8.8.1　ローテート命令の働き

ずらして押し出されたビットを最後の空いたビットに返して円環のような形式にするのを「ローテート」と言い，押し出されたままずらす形式を「シフト」と言います．

代表的なローテートとシフト命令を図で解説しましょう．

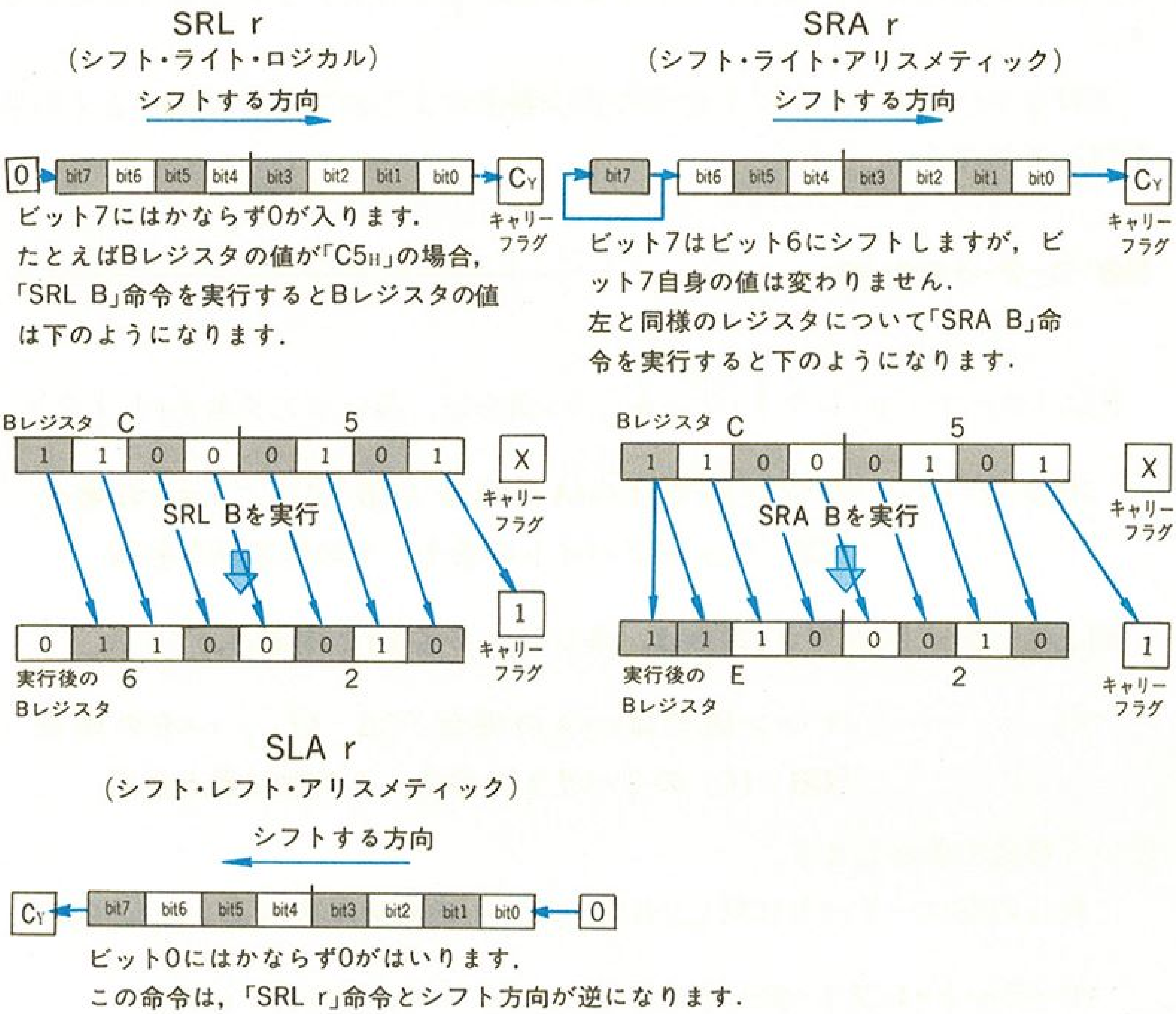
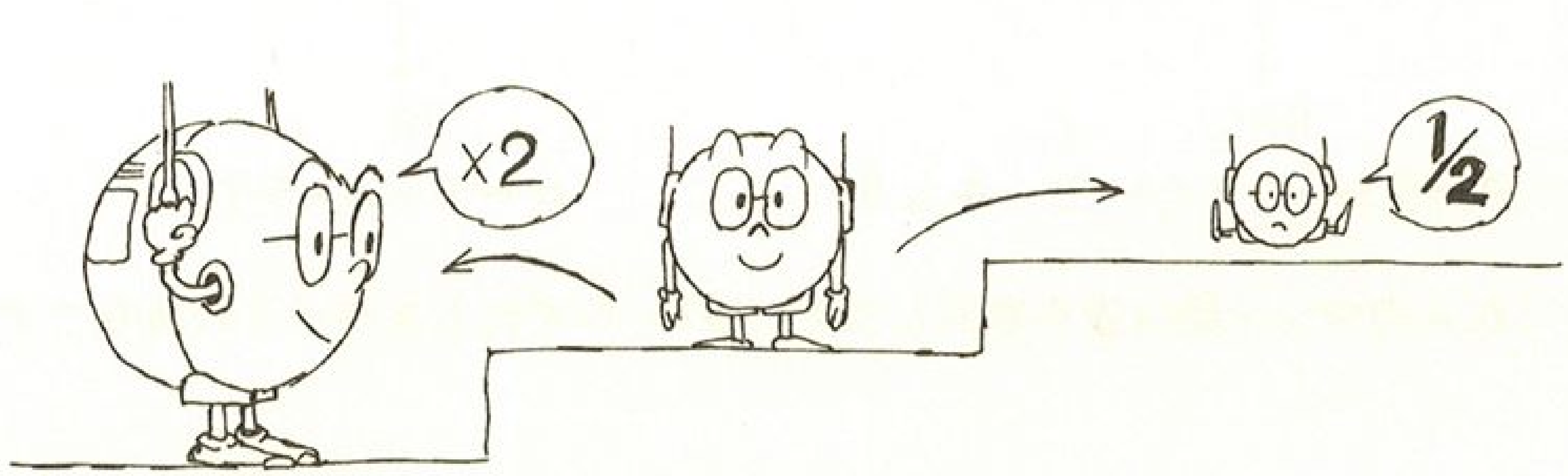


Figure-8.8.2 シフト命令の動き



ローテート，シフト命令は，このようにいろいろな種類がありますが，これらの命令の働きは様々なことに応用されています．この働きとは，たとえば「ローテートやシフトをしながらキャリーフラグを見ていれば，何番目のビットが0であるか1であるかを判別できる」ということや，「左にシフトすると値が2倍になり，右にシフトすると値が$\frac{1}{2}$になる」というようなことです．

主要なローテート，シフト命令の表は巻末にまとめて示してありますので参照してください．

●● ローテート命令 ●

RLC(ローテート・レフト・サーキュラ)命令は，各レジスタ名をrとすると，

RLC　r ………マシン語ではr=Aの場合「CB　07」，r=Bの場合「CB　00」の2バイトの命令．その他は表を参照．

RL(ローテート・レフト)命令は，各レジスタ名をrとすると，

RL　r ………マシン語ではr=Aの場合「CB　17」，r=Bの場合「CB　10」の2バイトの命令．その他は表を参照．

という形式で表わします．

これらの左ローテートに対し，右ローテートの命令は次のようになります．

ローテート・レフト・サーキュラ	ローテート・レフト
RLC　r	RL　r
↓	↓
RRC　r	RR　r
ローテート・ライト・サーキュラ	ローテート・ライト

これらのマシン語は表を参照してください．いずれも2バイトの命令です．

以上のローテート命令はいずれも2バイトの命令ですが，8080，8085CPUと互換性を持たせるために，Aレジスタに関するローテート命令に1バイト命令が用意されています．それを次に示しておきます．

RLC	A	と同機能の1バイト命令	⇒RLCA…マシン語では	「07」	ただし，これらの命令を実行することによるフラグの変化はC_Y（キャリー）フラグのみであることに注意．
RRC	A	〃	⇒RRCA…　〃	「0F」	
RL	A	〃	⇒RLA …　〃	「17」	
RR	A	〃	⇒RRA …　〃	「1F」	

これらはどちらを使っても自由ですが，フラグの変化のしかたが異なります．実行速度はこの命令の場合は，1バイト命令の方が速く，2バイト命令の$\frac{1}{2}$の時間で処理できます．

●● シフト命令 ●

SRL（シフト・ライト・ロジカル（論理的））命令は，各レジスタ名をrとすると，

SRL　r………マシン語ではr＝Aの場合「CB　3F」，r＝Bの場合「CB　38」の2バイトの命令．その他は表を参照．

SLA（シフト・レフト・アリスメティック（算術的））命令は，各レジスタ名をrとすると，

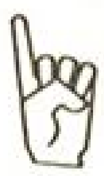
SLA　r………マシン語ではr＝Aの場合「CB　27」，r＝Bの場合「CB　20」の2バイトの命令．その他は表を参照．

SRA（シフト・ライト・アリスメティック）命令は，各レジスタ名をrとすると，

SRA　r………マシン語ではr＝Aの場合「CB　2F」，r＝Bの場合「CB　28」の2バイトの命令．その他は表を参照．

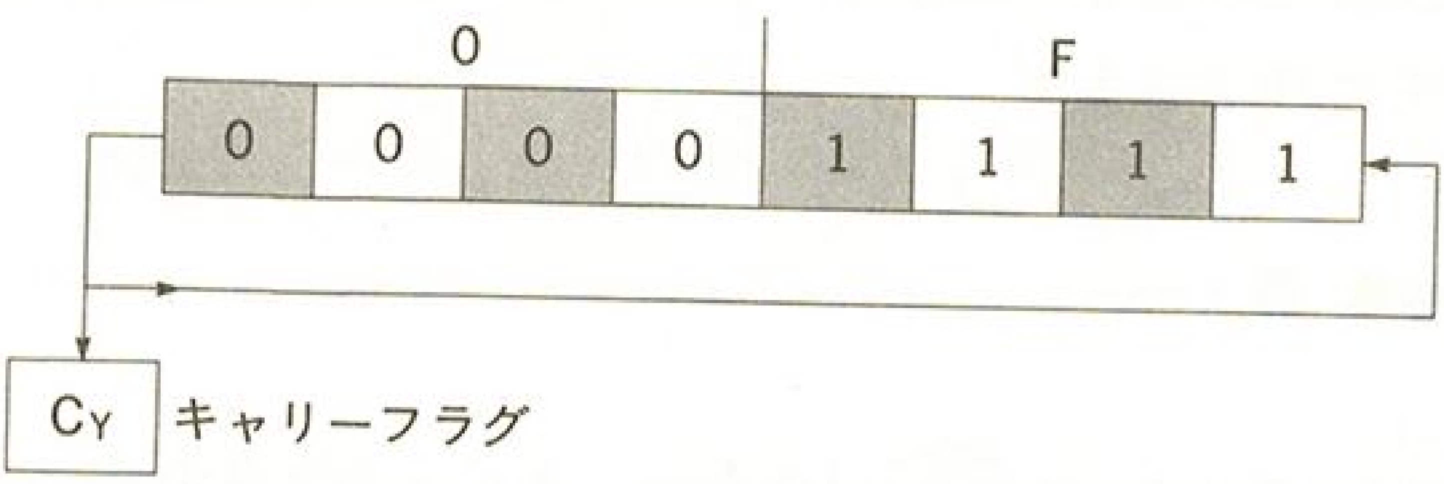
という形式で表わします．

算術的(アリスメティック)シフトというのは，本書では取り扱っていませんが，符号付き算術演算(最上位ビットを＋－の符号と考え，「0」のとき正，「1」のとき負として取り扱う．8ビットでは－128～＋127を表現できる)に利用する場合に，シフトしても符号が変化しないように考えられた命令です．



実習12 RLC命令(ローテート・レフト・サーキュラ)

左回転命令の動作を確認する単純な実習をしてみましょう．



上のように「0FH」のデータをAレジスタにロードしておき，Aレジスタに対してRLC命令を8回連続して実行し，その中間の4回目と最終のAレジスタの内容をメモリにストアして，あとで確認します．また，最終のキャリーフラグの状態も確認してみましょう．

このプログラムは特に難しい個所はありませんので，ハンド・アセンブルした結果を次に示します．

ここでは「RLC　A」命令として2バイトのオブジェクトにしていますが，これは前述の「RLCA」命令として1バイトのオブジェクトにしてもかまいません．

```
                                        ORG    9000H
9000H  3E  0F  ←─────────  LD     A,0FH  ………最初のデータ「0000
                                                   1111」をAレジスタ
                                                   にロード.
9002H  CB  07  ←─────────  RLC    A ┐
9004H  CB  07  ←─────────  RLC    A │ Aレジスタに対して左ローテート
9006H  CB  07  ←─────────  RLC    A │ を4回くり返す.
9008H  CB  07  ←─────────  RLC    A ┘
900AH  32  00  91  ←─────  LD     (9100H),A ……その結果をアドレス
                                                   9100Hのメモリにス
                                                   トア.
900DH  CB  07  ←─────────  RLC    A ┐
900FH  CB  07  ←─────────  RLC    A │
9011H  CB  07  ←─────────  RLC    A │ さらに4回くり返す.
9013H  CB  07  ←─────────  RLC    A ┘
9015H  32  01  91  ←─────  LD     (9101H),A ……その結果をアドレス
                                                   9101Hのメモリにス
                                                   トア.
9018H  DA  21  90  ←─────  JP     C,[キャリー「1」] …現在もしキャリーが
                                                   「1」であれば[キャリー「1」]
                                                   にジャンプ.
901BH  3E  00  ←─────────  LD     A,00      ┐ キャリーが「0」の場合
901DH  32  02  91  ←─────  LD     (9102H),A ┘ アドレス9102Hには
                                                   「00」をストア.
9020H  C9  ←─────────────  RET    ………………当サブルーチンを終了
                                                   してMCモニタに戻る.
9021H  3E  01  ←[キャリー「1」]  LD     A,01      ┐ キャリーが「1」の場合
9023H  32  02  91  ←─────  LD     (9102H),A ┘ アドレス9102Hに「01」
                                                   をストア.
9026H  C9  ←─────────────  RET    ………………当サブルーチンを終
                                                   了してMCモニタに
                                                   戻る.
```

このオブジェクト・プログラムはアドレス9000_Hからメモリに書き込みます．データ「$0F_H$」に対してRLC命令を4回行なった結果は，アドレス9100_Hに，8回行なった結果は9101_Hに，そして最終のキャリーフラグの状態は9102_Hにストアされています．キャリーフラグの状態は，キャリーフラグが「0」であれば「00」が，「1」であれば「01」がストアされます．

ではこの「実習12」のプログラムをMCモニタでメモリに書き込み，実行して結果を確認してみましょう．その実行例を次に示します．

```
-S9000 ↵ ………Sコマンドでオブジェクト・プログラムを書き込む.
9000 47 3E ↵
9001 F0 0F ↵
9002 A1 CB ↵
9003 D3 07 ↵
9004 10 CB ↵
9005 F3 07 ↵
9006 3A CB ↵
9007 01 07 ↵
9008 F0 CB ↵
  ：  (途中省略)
901F 5D 91 ↵
9020 07 C9 ↵
9021 00 3E ↵
9022 AF 01 ↵
9023 21 32 ↵
9024 3E 02 ↵
9025 FF 91 ↵
9026 21 C9 ↵
9027 3B .  ↵

-D9000 9026 ↵ ………ダンプして確認.
9000   3E 0F CB 07 CB 07 CB 07 CB 07 32 00 91 CB 07 CB
9010   07 CB 07 CB 07 32 01 91 DA 21 90 3E 00 32 02 91
9020   C9 3E 01 32 02 91 C9

-S9100 ↵ ………結果がストアされるメモリを0クリアしておく.
9100 E6 00 ↵
9101 01 00 ↵
9102 28 00 ↵
9103 FA 00 ↵
9104 3A .  ↵

-G9000 ↵ ………実習12のプログラムを実行する. スタートアドレスは9000H.

-D9100 9103 ↵ ………実行結果の確認.
9100   F0 0F 01 00
        ↗   ↖   ↖ CYフラグは「1」
_
4ビットずれている.  もとに戻った.
```

Figure-8.8.3 実習12(RLC命令)の実行

実習13 SLA命令(シフト・レフト・アリスメティック)

左シフト命令を使って，数値を2倍にする単純な実習をしてみましょう．データ「19_H」(10進の「25」)をAレジスタにロードしておき，SLA命令を実行

して，Aレジスタの結果を確認してみます．

このプログラムのハンド・アセンブルした結果を次に示します．

```
                                   ORG   9000H
9000H  3E  19      ←――――――――――      LD    A ,19H   ………値19H＝25をAレジス
                                                        タにロード．
9002H  CB  27      ←――――――――――      SLA   A ………………左シフト．
9004H  32  00  91  ←――――――――        LD    (9100H),A ……その結果をアドレス91
                                                        00Hのメモリにストア．
9007H  C9      ←――――――――――――――      RET   ………………当サブルーチンを終了
                                                        してMCモニタに戻る．
```

このオブジェクト・プログラムをMCモニタでアドレス9000_Hからメモリに書き込み，実行して結果を確認した例を次に示します．「19_H＝25」がSLA命令により2倍にされて「32_H＝50」になっていることがわかります．

シフトやローテート命令をうまく利用すると，加算や減算命令だけをくり返し使うよりも実行速度の速い掛け算や割り算の演算ルーチンを作ることが可能です．

```
-S9000
9000 47 3E    ┐
9001 F0 19    │
9002 A1 CB    │
9003 D3 27    │ Sコマンドでオブジェクト・プログラムを書き込む．
9004 10 32    │
9005 F3 00    │
9006 3A 91    │
9007 01 C9    ┘
9008 F0 .     ………Sコマンドを終わる．

-D9000 9007 ………ダンプして確認．
9000  3E 19 CB 27 32 00 91 C9

-S9100 ………結果がストアされるメモリを0クリアしておく．
9100 E6 00
9101 01 00
9102 28 .

-G9000 ………実習13のプログラムを実行する．

-D9100 9101 ………実行結果の確認．
9100  32 00
      ↑
-     19H × 2 = 32H (25 × 2 = 50)
```

Figure-8.8.4 実習13(SLA命令)の実行

8.9

●16ビット・ロード命令

「16ビット・ロード命令」とは，8.2章で解説した「8ビット・ロード命令」が各レジスタを8ビット単位で操作する命令であるのに対し，

Bレジスタと Cレジスタを組(ペア)にして ⇒ 16ビットのBCレジスタペア
Dレジスタと Eレジスタを組にして ⇒ 16ビットのDEレジスタペア
Hレジスタと Lレジスタを組にして ⇒ 16ビットのHLレジスタペア
Aレジスタとフラグ・レジスタを組にして ⇒ 16ビットのAFレジスタ

として，16ビット単位（2バイト）でレジスタ↔メモリ間のロード，ストアを操作する命令です．Figure-7.2.1のZ-80CPUの内部レジスタの図を参照してください．

また，各レジスタペアの内容とスタックエリアとの退避／復帰命令である「PUSH」,「POP」命令については，8.7章でも解説していますので参照してください．

巻末に主要な16ビット・ロード命令の表を示します．なお，「エクスチェンジ命令」に属する命令ですが，DEレジスタペアの内容と，HLレジスタペアの内容をそっくり交換する命令などもありますのでここに紹介しておきます．

EX　DE, HL………マシン語では「EB」の1バイトの命令．

このレジスタ交換(エクスチェンジ)命令は実習しませんが，たとえばこの命令を連続して

2度実行すると，各レジスタはもとの内容に戻り，何も実行しなかったのと同じ結果になるわけです．

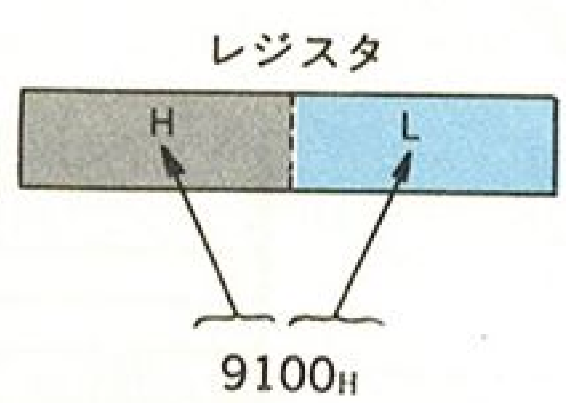
ここまで実習が進むと，命令表を見るだけでも各命令の働きをある程度推測することができるのではないかと思います．

さっそく実習にはいりましょう．

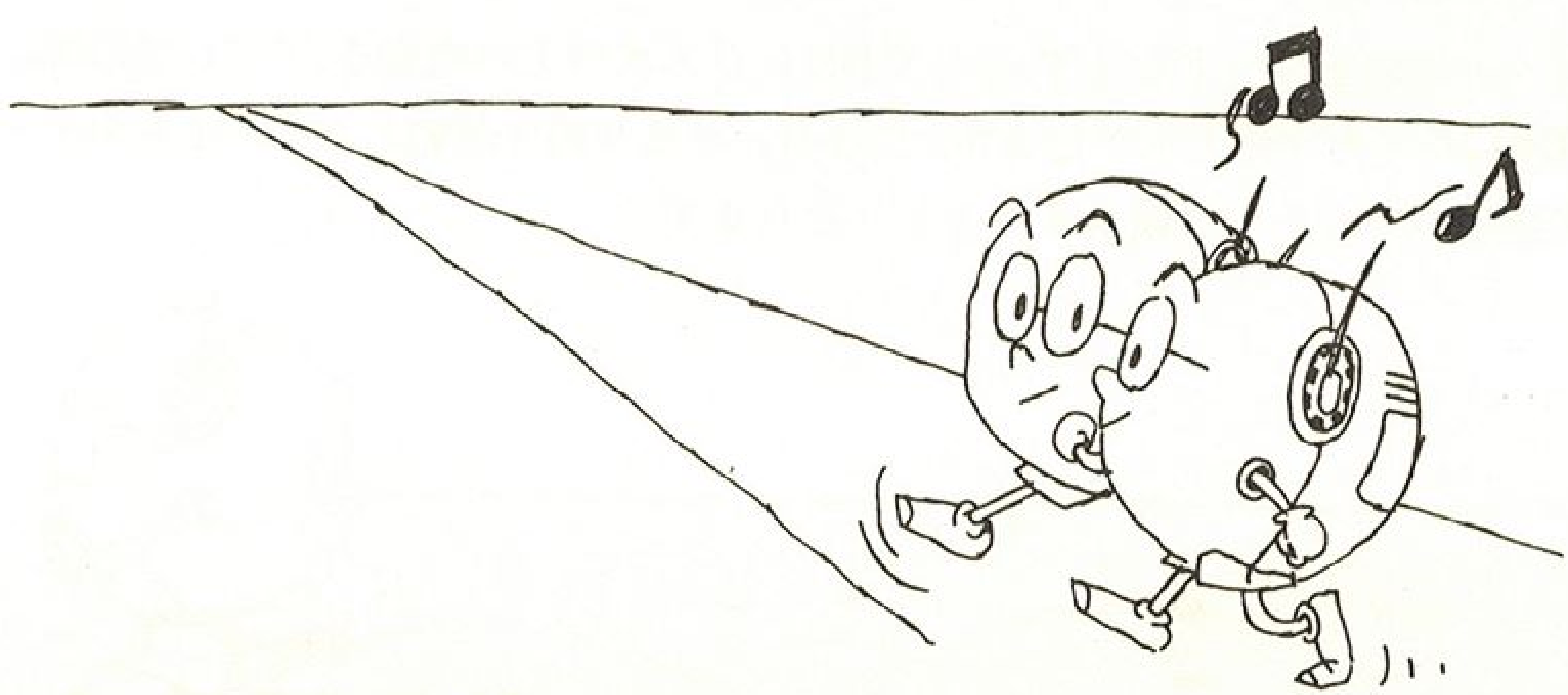
実習14 16ビットデータのロード

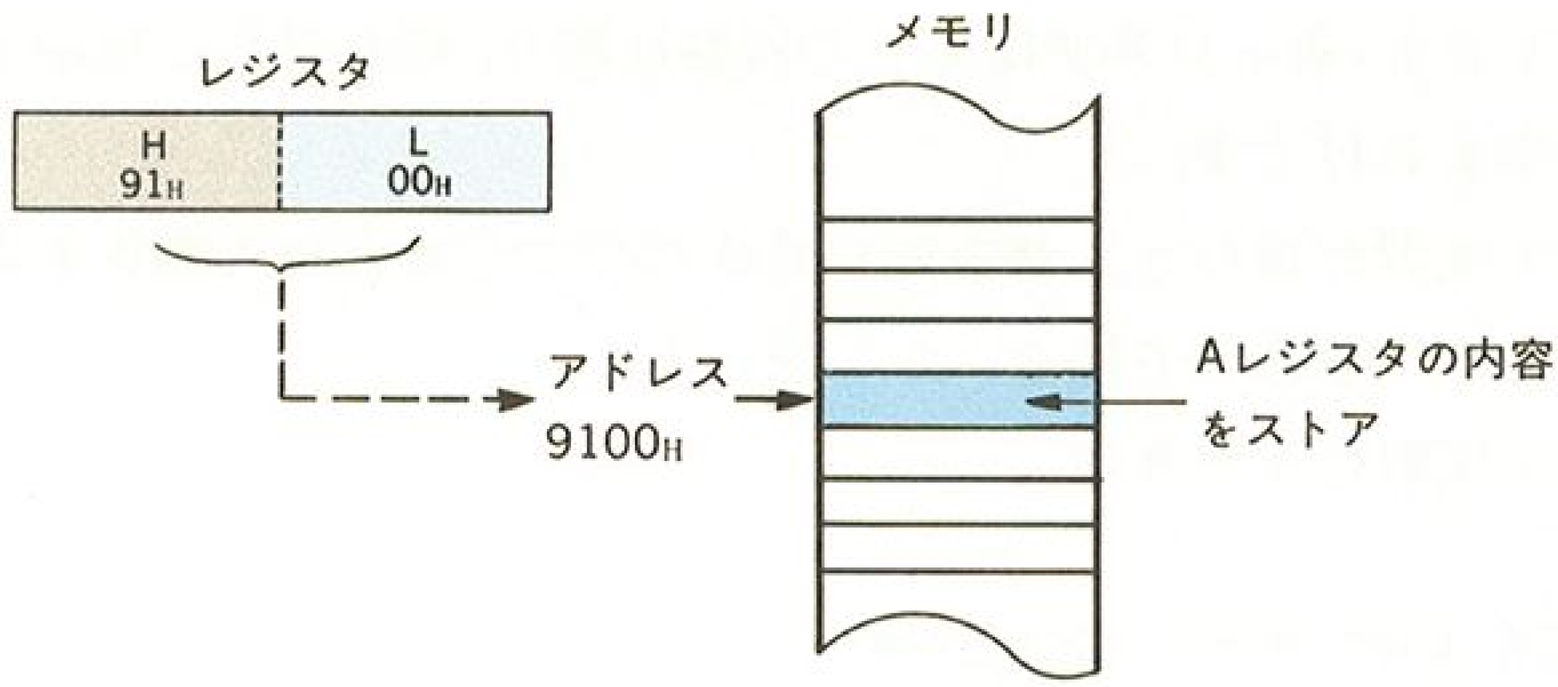
16ビット単位(連続した2バイト)のデータのやり取りの実習を行ないましょう．

まず，HLレジスタペアに16ビット(2バイト)の値9100_Hをロードします．

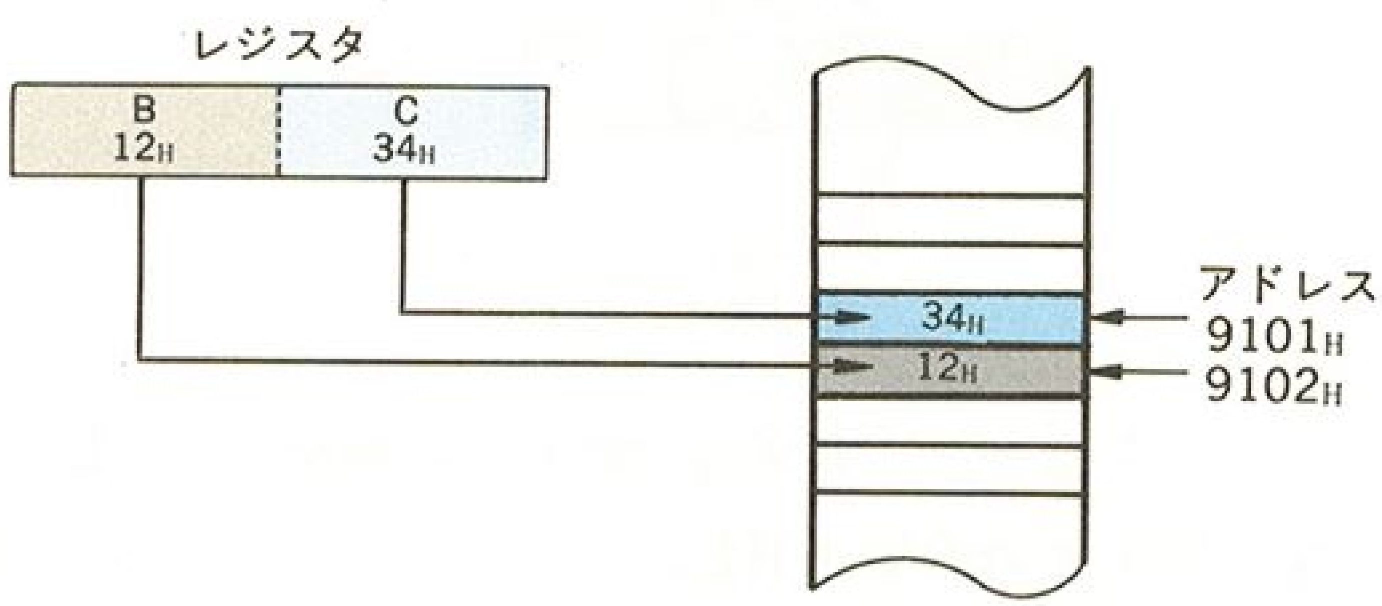


次に8ビット・ロード命令の「実習4」ですでに実習した「LD (HL),A」命令を使って，Aレジスタの内容をHLレジスタペアがアドレスするメモリにストアします．



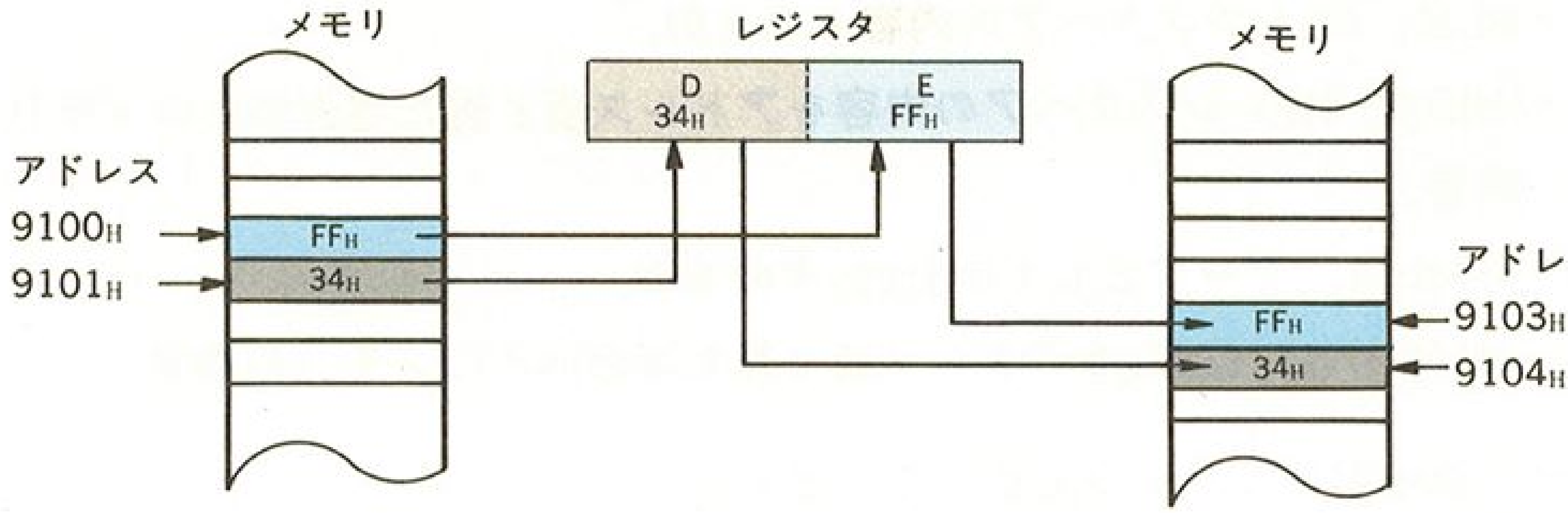


次は，BCレジスタペアに16ビットの値1234_Hをロードします．そして，そのBCレジスタペアの内容(2バイト)を，アドレス9101_Hとその次のメモリにストアします．



最後は，DEレジスタペアにアドレス9100_H，9101_Hにストアされている16ビットの値をロードします．そしてDEレジスタペアの内容を，アドレス9103_H，9104_Hのメモリにストアします．つまり，メモリ内で連続した2バイトのデータを別のアドレスに転送することになります．





以上，内容的には3つの部分に分けられますが，このプログラムをハンド・アセンブルして，その結果を次に示します．

```
                                      ORG  9000H
9000H  21 00 91 ←──── LD   HL,9100H  ……HLレジスタペアに値9100Hをロード.
9003H  3E FF ←─────── LD   A,0FFH    ………Aレジスタに値FFHをロード.
9005H  77 ←────────── LD   (HL),A    ………HLレジスタペアの内容がアドレス
                                                するメモリにAレジスタの内容を
                                                ストア(これは8ビット扱い).
9006H  01 34 12 ←──── LD   BC,1234H  ……BCレジスタペアに値1234Hをロード.
9009H  ED 43 01 91 ← LD   (9101H),BC …BCレジスタペアの内容をアドレス
                                                9101H, 9102Hにストア.
900DH  ED 5B 00 91 ← LD   DE,(9100H) …DEレジスタペアにアドレス9100H
                                                と9101Hの内容をロード.
9011H  ED 53 03 91 ← LD   (9103H),DE …DEレジスタペアの内容をアドレス
                                                9103H, 9104Hのメモリにストア.
9015H  C9 ←────────── RET            ………………当サブルーチンを終了してMCモ
                                                ニタに戻る.
```

このプログラムで注意しなければならないのは，

HLと(HL)

9100Hと(9100H)

などの(　)が付いている場合と，付いていない場合の区別をしっかりと理解しておくことです．実習2でも解説しましたが，ここは実に間違いやすいところで，相当な経験者でも時として勘違いをしてしまう部分です．

・HLは，HLレジスタペアの内容そのもの．

・(HL)は，HLレジスタペアの内容をアドレス値と見た場合のそのメモリの内容．

・9100Hは，データとしての9100$_H$そのもの．

・(9100H)は，9100$_H$をアドレス値と見た場合のそのメモリの内容．

という意味なので，十分注意してください．

ではこのオブジェクト・プログラムをMCモニタでメモリに書き込み，実行結果を確認してみましょう．

```
-S9000↵ ………Sコマンドでオブジェクト・プログラムを書き込む．
9000 47 21↵
9001 F0 00↵
9002 A1 91↵
9003 D3 3E↵
9004 10 FF↵
9005 F3 77↵
9006 3A 01↵
9007 01 34↵
9008 F0 12↵
9009 E6 ED↵
900A FE 43↵
900B D3 01↵
900C 40 91↵
900D F6 ED↵
900E 01 5B↵
900F D3 00↵
9010 40 91↵
9011 FB ED↵
9012 C9 53↵
9013 5B 03↵
9014 50 91↵
9015 52 C9↵
9016 4E .↵

-D9000 9015↵ ………ダンプして確認．
9000  21 00 91 3E FF 77 01 34 12 ED 43 01 91 ED 5B 00
9010  91 ED 53 03 91 C9

-S9100↵ ………結果がストアされるメモリを0クリアしておく．
9100 E6 00↵
```

```
9101 01 00
9102 28 00
9103 FA 00
9104 3A 00
9105 48 00
9106 F0 .

-G9000 ………実習14のプログラムを実行する.

-D9100 9105 ………実行結果の確認.
9100  FF 34 12 FF 34 00
-        1234Hをストア.
FFHをストア   2バイトの転送
```

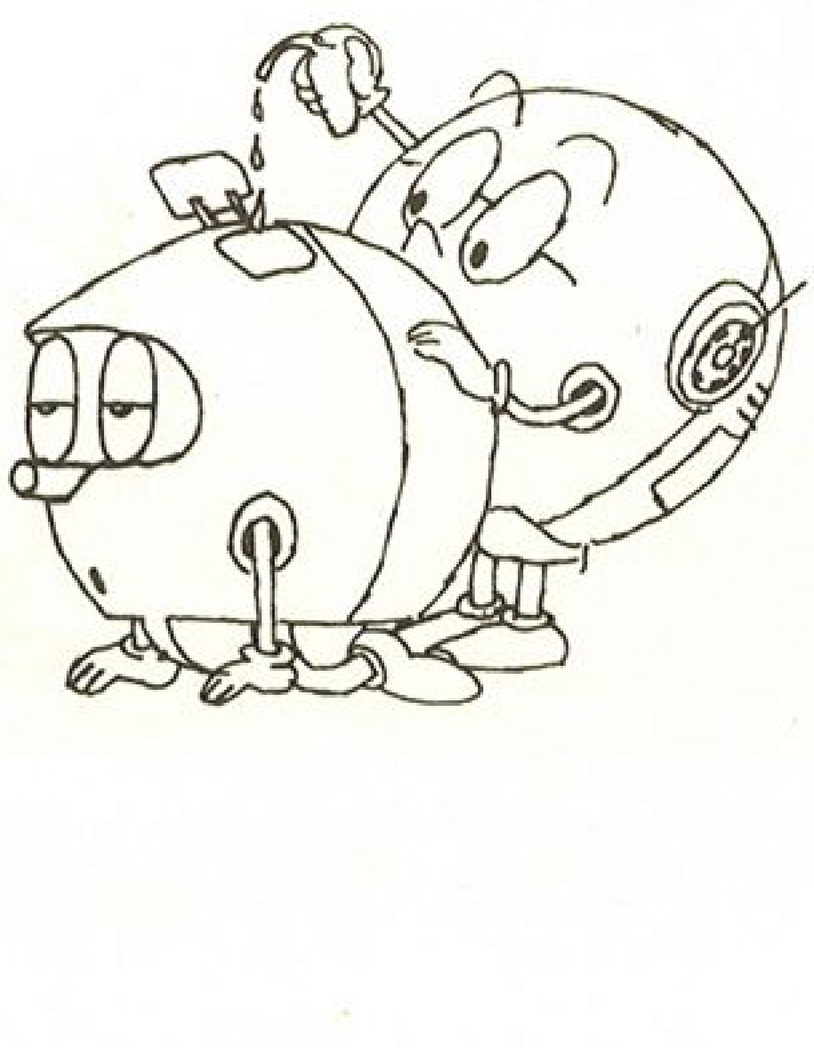
Figure-8.9.1 実習14(16ビットデータのロード)の実行

実習15 PUSHとPOP

スタック操作命令でもあるPUSH命令とPOP命令については, 8.7章に解説がありますので参照してください.

ここではこれらの命令の動作を確認するだけの単純なプログラムを作って実習しましょう. 次のようなプログラムを作ります.

HLレジスタペアにある適当な16ビットの値をロードしておきます. それをスタックに退避(PUSH)したあと, 同じHLレジスタペアに別の値をロードし, そのときのHLレジスタペアの内容をメモリにストアしておきます. 次に, スタックに退避しておいたもとのデータをHLレジスタペアに復帰(POP)して, そのときのHLレジスタペアの内容をメモリにストアし, 前のものと比較してPUSH, POPの動作を確認しようというものです.

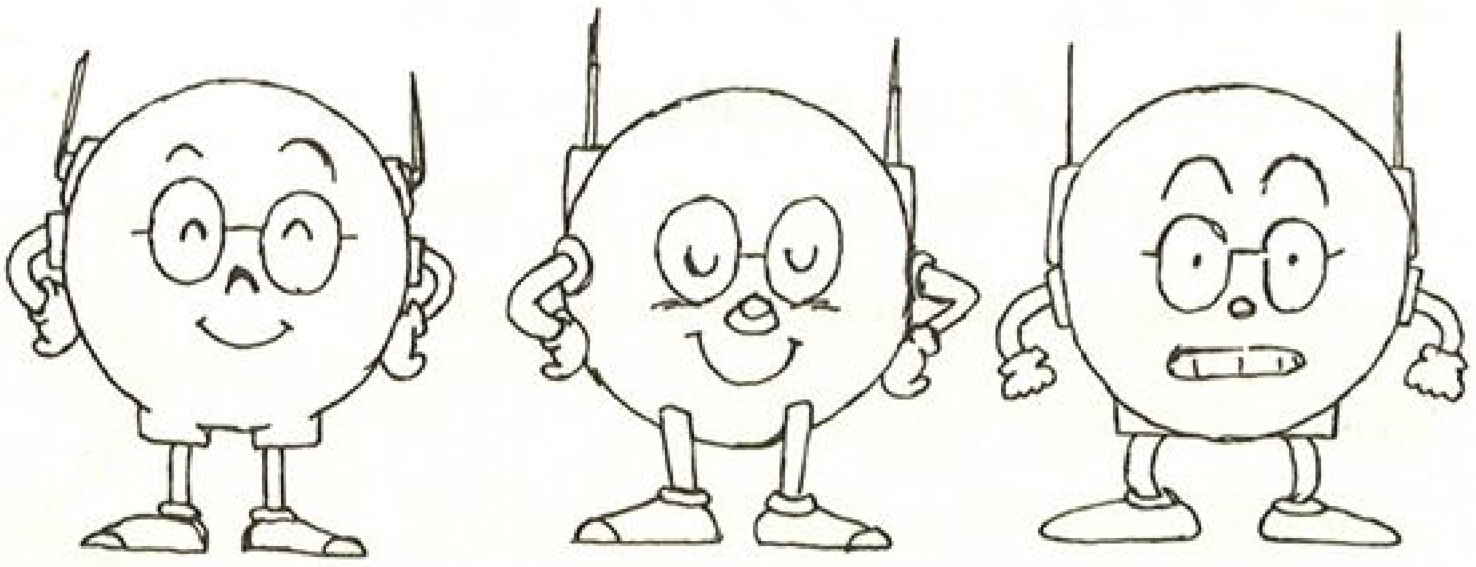


簡単なプログラムなので, ハンド・ア

センブルした結果を次に示します.

アドレス	機械語			ニーモニック	オペランド	説明
				ORG	9000H	
9000_H	21	34	12	LD	HL ,1234H	……HLレジスタペアに値1234_Hをロード.
9003_H	E5			PUSH	HL	……HLレジスタペアの内容をスタックに退避.
9004_H	21	78	56	LD	HL ,5678H	……HLレジスタペアに別の値5678_Hをロード.
9007_H	22	00	91	LD	(9100H), HL	……そのHLレジスタペアの内容をアドレス9100_Hのメモリにストア(2バイト).
$900A_H$	E1			POP	HL	……もとの値をスタックからHLレジスタペアに復帰.
$900B_H$	22	02	91	LD	(9102H), HL	……そのHLレジスタペアの内容をアドレス9102_Hのメモリにストア(2バイト).
$900E_H$	C9			RET		……当サブルーチンを終了してMCモニタに戻る.

ではこのオブジェクト・プログラムをMCモニタでメモリに書き込み，実行して結果を確認してみましょう．その実行例を次に示します.



```
-S9000 ↵………Sコマンドでオブジェクト・プログラムを書き込む.
9000 21 21 ↵
9001 00 34 ↵
9002 91 12 ↵
9003 3E E5 ↵
9004 FF 21 ↵
9005 77 78 ↵
9006 01 56 ↵
9007 34 22 ↵
9008 12 00 ↵
9009 ED 91 ↵
900A 43 E1 ↵
900B 01 22 ↵
900C 91 02 ↵
900D ED 91 ↵
900E 5B C9 ↵
900F 00 .  ↵

-D9000 900E ↵………ダンプして確認.
9000  21 34 12 E5 21 78 56 22 00 91 E1 22 02 91 C9
-S9100 ↵………結果がストアされるメモリを0クリアしておく.
9100 FF 00 ↵
9101 34 00 ↵
9102 12 00 ↵
9103 FF 00 ↵
9104 34 00 ↵
9105 00 .  ↵

-G9000 ↵………実習15のプログラムを実行する.

-D9100 9104↵………実行結果の確認.
9100  78 56 34 12 00
          ↑     ↑
-   同じレジスタ  PUSH, POPした値
    にロードした
    別の値.
```

Figure-8.9.2 実習15(PUSHとPOP)の実行

8.10

●16ビット算術演算命令

16ビット算術演算命令は，主にHLレジスタペアと，各レジスタペアの間の加算および減算，それと各レジスタペアおよびSPの値の＋１および－１の演算機能を持った命令群です．その主要命令の表は巻末に示してありますので参照してください．

なお，レジスタペア間の加算，減算命令は，多桁演算を行なう場合を考慮して，キャリーフラグ(C_Y)の値を含めた演算ができるようになっています．ただし，本書ではこの機能に関する詳しい実習は行ないません．

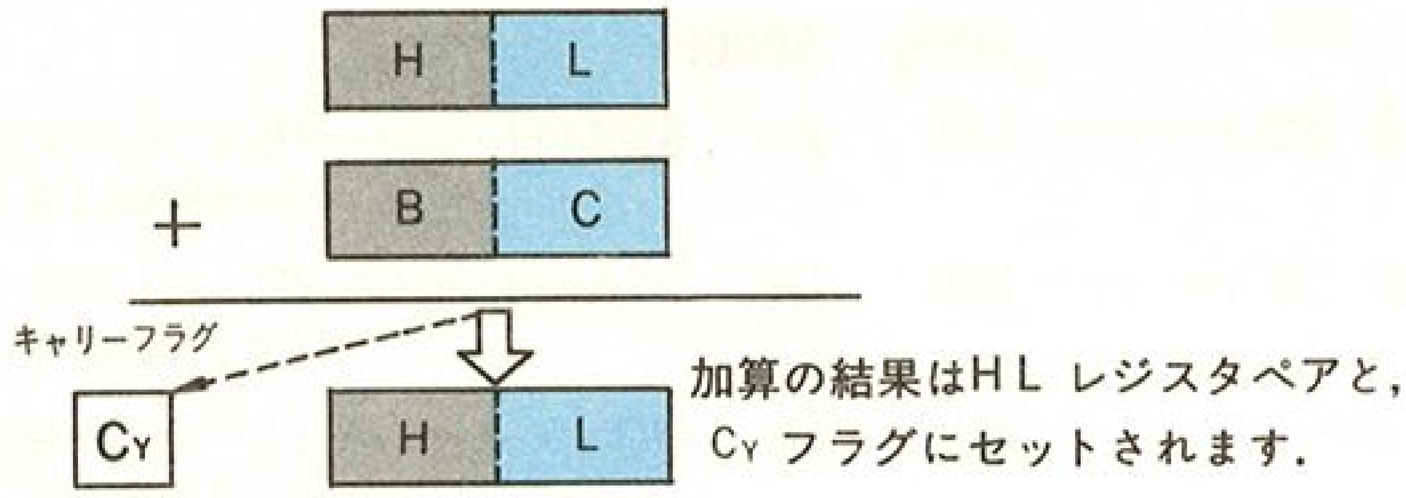
ではさっそく16ビットの加算，減算の実習から始めましょう．

実習16 16ビットデータの加減算

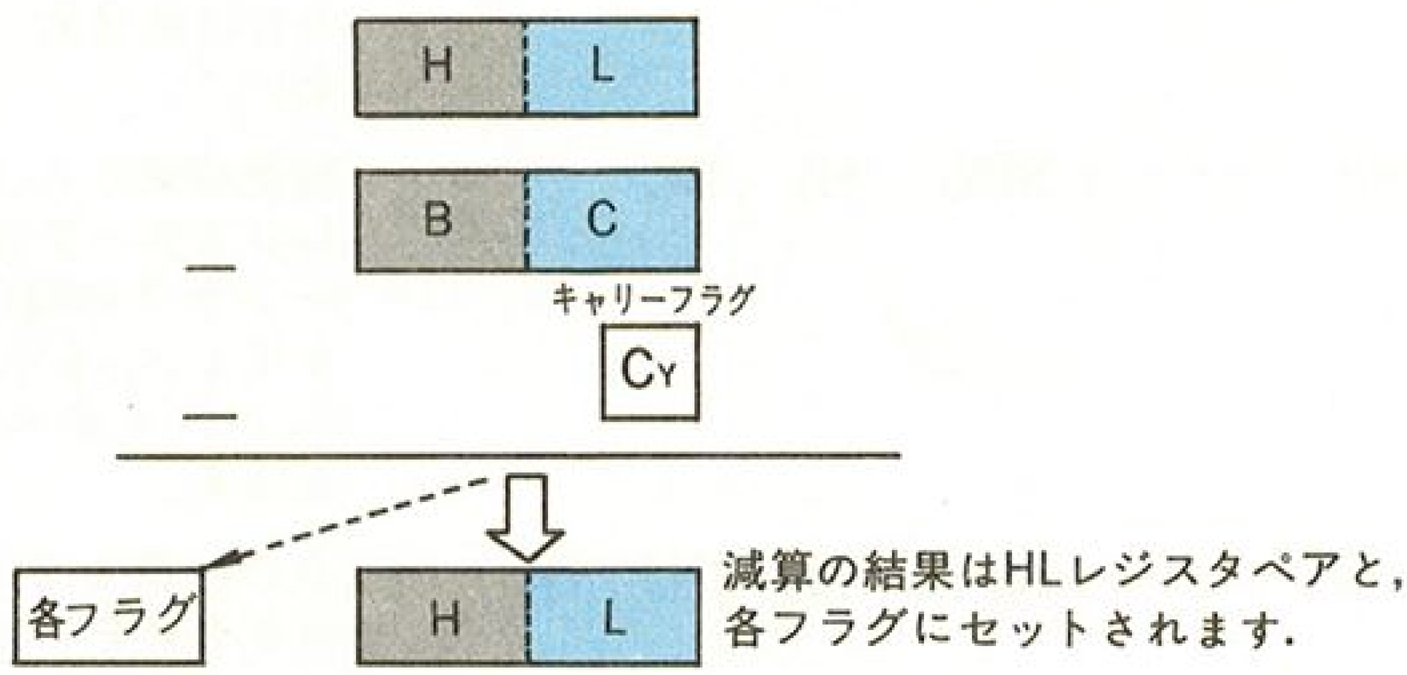
単純な16ビットの加算命令と減算命令を実習してみましょう．

まずHLレジスタペアには適当な値として35750(＝$8BA6_H$)をロードしておき，BCレジスタペアには14250(＝$37AA_H$)をロードします．「HL＋BC」の演算を行ない，次に「HL－BC」の演算（HLレジスタペアには「HL＋BC」の結果がセットされている)を行ないます．それぞれの結果をメモリにストアします．それをあとで確認してみましょう．

「HL＋BC」の命令は「ADD　HL ，BC」であり，次のような加算を行ないます．



「HL－BC」の命令は「SBC　HL ,BC」であり，次のような減算を行ないます．



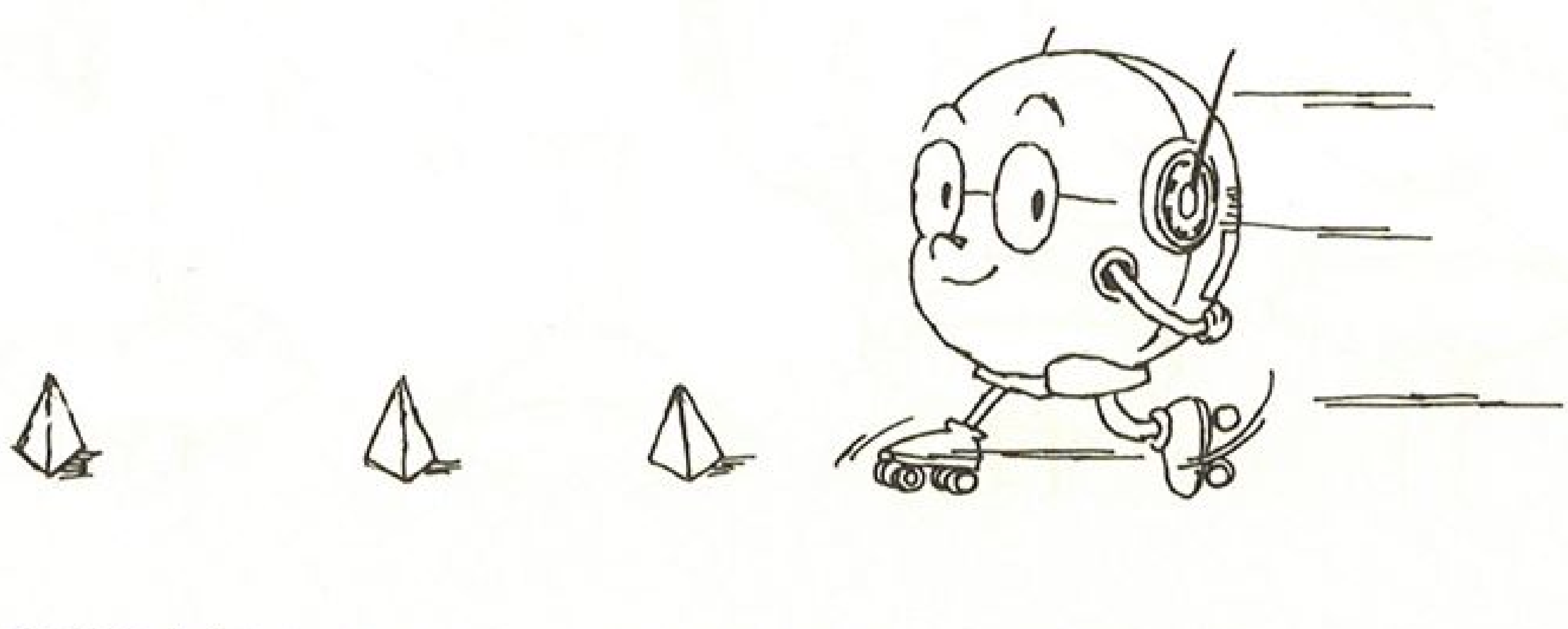
そのプログラムのハンド・アセンブルした結果を次に示します．



```
                                    ORG   9000H
9000H  21  A6  8B ←───────  LD    HL ,8BA6H ·········HLレジスタペアに値35750
                                                     (＝8BA6H)をロード.
9003H  01  AA  37 ←───────  LD    BC ,37AAH ·········BCレジスタペアに値14250
                                                     (＝37AAH)をロード.
9006H  09 ←───────────────  ADD   HL ,BC  ············HL＋BCの演算.結果はHL
                                                     レジスタペアに.
9007H  22  00  91 ←───────  LD    (9100H), HL ······その内容をアドレス9100H
                                                     ～1のメモリにストア.
900AH  B7 ←───────────────  OR    A  ··················キャリーフラグを「0」にす
                                                     るだけの目的. 他に意味は
                                                     ない.*
900BH  ED  42 ←───────────  SBC   HL ,BC  ············加算結果から,加算したBC
                                                     レジスタペアの値とキャリ
                                                     ーフラグの値(上の命令で
                                                     0にした)を引く. もとの
                                                     HLレジスタペアの値にな
                                                     るはず.
900DH  22  02  91 ←───────  LD    (9102H), HL ······その結果をアドレス9102H
                                                     ～3のメモリにストア.
9010H  C9 ←───────────────  RET   ························当サブルーチンを終了して
                                                     MCモニタに戻る.
```

このオブジェクト・プログラムをMCモニタでメモリに書き込み，実行して結果を確認しましょう．その実行例を次に示します．

なお，16ビットで表現可能な符号なしの数値は0～65535（0_H～$FFFF_H$）ですのでHLやBCレジスタペアの値を変えて試みるときは注意してください．



*OR命令やAND命令を実行すると，いかなる場合でもキャリーフラグは「0」になる．

```
-S9000 ↵ ………Sコマンドでオブジェクト・プログラムを書き込む.
9000 21 21 ↵
9001 34 A6 ↵
9002 12 8B ↵
9003 E5 01 ↵
9004 21 AA ↵
9005 78 37 ↵
9006 56 09 ↵
9007 22 22 ↵
9008 00 00 ↵
9009 91 91 ↵
900A E1 B7 ↵
900B 22 ED ↵
900C 02 42 ↵
900D 91 22 ↵
900E FF 02 ↵
900F 00 91 ↵
9010 91 C9 ↵
9011 ED . ↵

-D9000 9010 ↵ ………ダンプして確認.
9000  21 A6 8B 01 AA 37 09 22 00 91 B7 ED 42 22 02 91
9010  C9

-S9100 ↵ ………結果がストアされるメモリを0クリアしておく.
9100 78 00 ↵
9101 56 00 ↵
9102 34 00 ↵
9103 12 00 ↵
9104 00 00 ↵
9105 00 . ↵

-G9000 ↵ ………実習16のプログラムを実行する.

-D9100 9104 ↵ ………実行結果の確認.
9100  50 C3 A6 8B 00
         ↑     ↑
-   C350H＝50000   8BA6H＝35750
```

Figure-8.10.1 実習16(16ビットデータの加減算)の実行

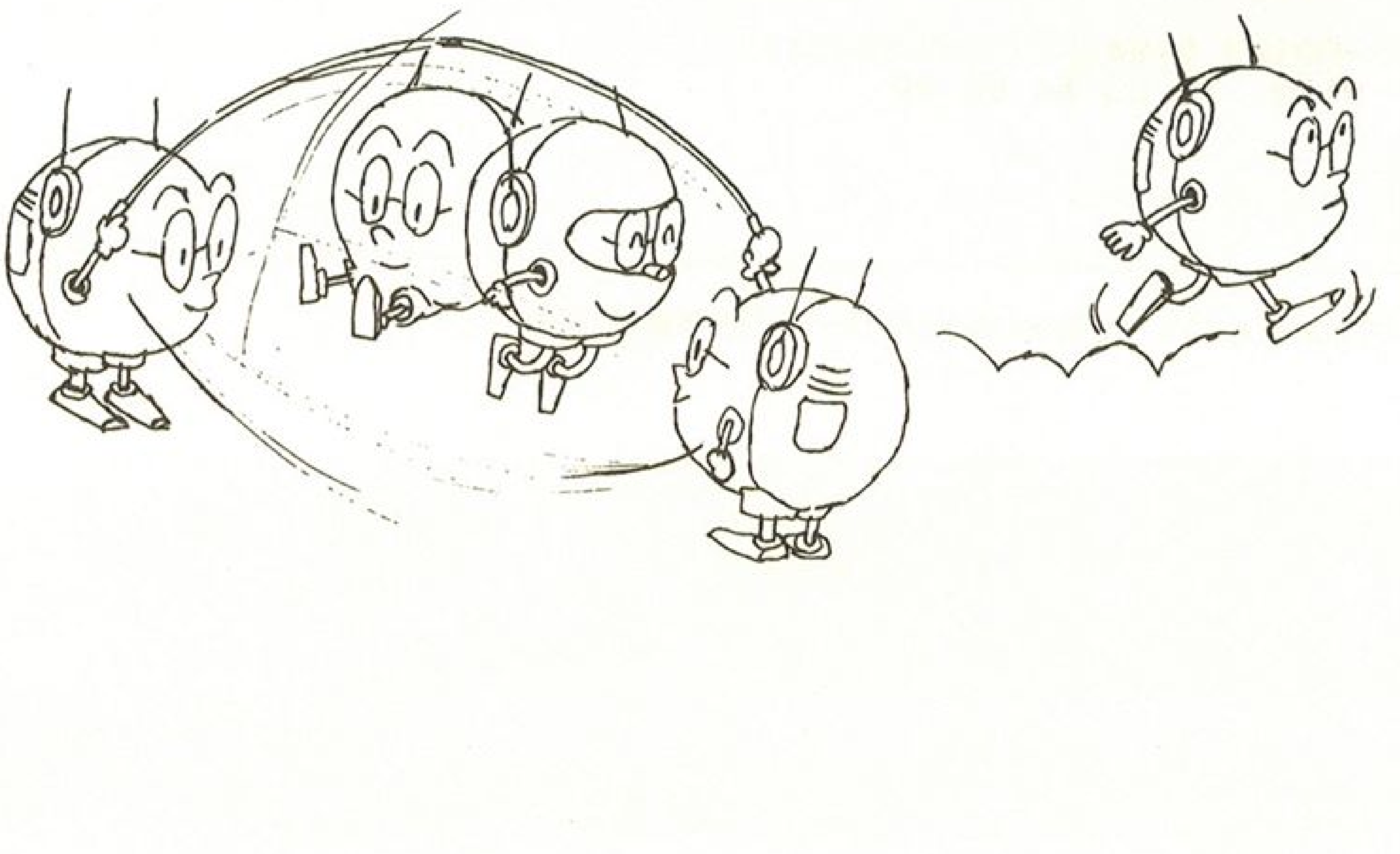
実習17　16ビットデータのインクリメント,デクリメント

各レジスタペアのインクリメント(＋１する)とデクリメント(－１する)の命令の実習です．各レジスタペア名をrrとすると，

インクリメントは　INC　rr………マシン語ではrr＝BCの場合「03」，
rr＝DEの場合「13」．その他は表を参照．
デクリメントは　DEC　rr………マシン語ではrr＝BCの場合「0B」，
rr＝DEの場合「1B」．その他は表を参照．

という形式で表わします．

ここでは，１章のMCモニタの実動テストに使ったFigure-1.2.5に示されている11バイトのプログラムを実習します．このプログラムは，アドレス9100_Hから256バイト，つまり9100_H～$91FF_H$の間のメモリに，00_Hから始まって01_H，02_H，03_H，……FE_H，FF_Hという計256バイトのデータを順次書き込んでいくものです．実はこのプログラムは，HLレジスタペアに対するインクリメント命令(INC)を使っていたのです．



ではこのプログラムのハンド・アセンブルした結果を次に示します.

```
                          ORG  9000H
9000H  AF ←————————————— XOR  A ……………Aレジスタを「00H」にするだけの
                                       目的*.
9001H  21 00 91 ←——————— LD   HL,9100H ……HLレジスタペアに値9100Hをロード
9004H  77 ←——[ループ]    LD   (HL),A ……HLレジスタペアの内容が示すアドレスに,Aレジスタの内容をストア.
9005H  23 ←————————————— INC  HL ……………HLレジスタペアを+1.
9006H  3C ←————————————— INC  A ……………Aレジスタを+1.
9007H  C2 04 90 ←——————— JP   NZ,[ループ]…Z(ゼロ)フラグが「1」でなければ[ループ]へジャンプしてループ**.
900AH  C9 ←————————————— RET  ……………当サブルーチンを終了してMCモニタに戻る.
```

このプログラムは「XOR」命令を除き, 今までに登場した命令の組合せですので, プログラムを見ただけでも何をやっているかがわかるものと思います.

このオブジェクト・プログラムをMCモニタでメモリに書き込み, 実行すれば, 1章のFigure-1.2.6, 1.2.7の結果と同じになるわけです.

DEC(デクリメント)命令を実行するには,

```
LD  HL ,9100H  →  LD  HL, 92FFH ………マシン語は「21 FF 92」
INC  HL  →  DEC  HL  ……………………マシン語は「2B」
```

に変更して実行し, アドレス9200H～92FFHを調べてみてください. この実行例を示すのは省略しますが, 結果は次のようになります.

＊Aレジスタとの XOR（エクスクルーシブ・オア：排他的論理和）を実行すると, Aレジスタの値が「00H」となる. 1バイト命令でAレジスタを0クリアできるのでよく使われる.

＊＊カウントアップしていき, FFHにさらに1加えたときにFFH→00Hとなることにより, Zフラグが「1」となる.

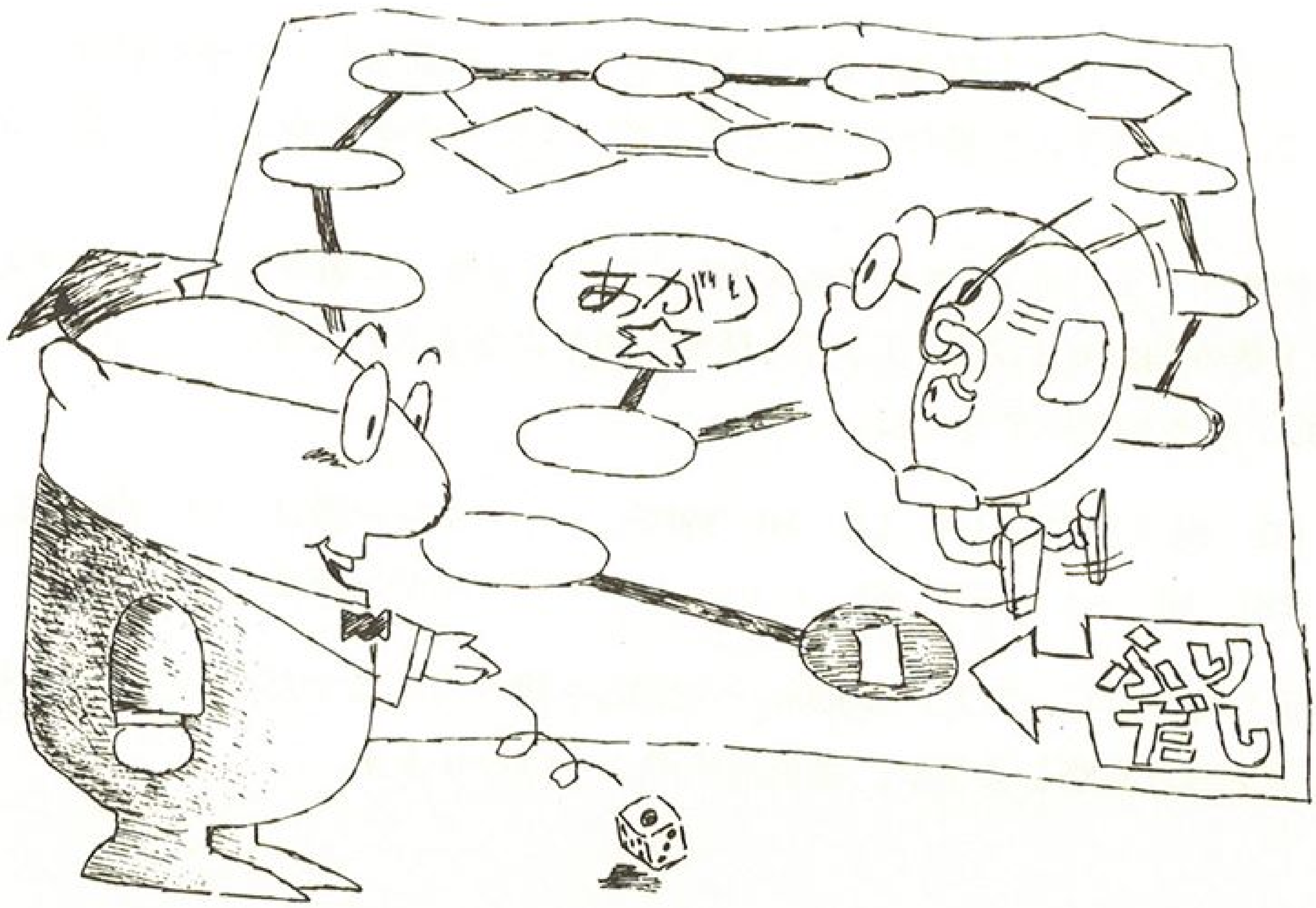
最後にストアされたデータ．

```
-D9200,92FF
9200  FF FE FD FC FB FA F9 F8 F7 F6 F5 F4 F3 F2 F1 F0
9210  EF EE ED EC EB EA E9 E8 E7 E6 E5 E4 E3 E2 E1 E0
9220  DF DE DD DC DB DA D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
9230  CF CE CD CC CB CA C9 C8 C7 C6 C5 C4 C3 C2 C1 C0
9240  BF BE BD BC BB BA B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
9250  AF AE AD AC AB AA A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
9260  9F 9E 9D 9C 9B 9A 99 98 97 96 95 94 93 92 91 90
9270  8F 8E 8D 8C 8B 8A 89 88 87 86 85 84 83 82 81 80
9280  7F 7E 7D 7C 7B 7A 79 78 77 76 75 74 73 72 71 70
9290  6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60
92A0  5F 5E 5D 5C 5B 5A 59 58 57 56 55 54 53 52 51 50
92B0  4F 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41 40
92C0  3F 3E 3D 3C 3B 3A 39 38 37 36 35 34 33 32 31 30
92D0  2F 2E 2D 2C 2B 2A 29 28 27 26 25 24 23 22 21 20
92E0  1F 1E 1D 1C 1B 1A 19 18 17 16 15 14 13 12 11 10
92F0  0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00
-
```

最初にストアされたデータ．

ここからアドレスは－1ずつ，データは＋1ずつされていく．

Figure-8.10.2　実習17のプログラムを一部変更してDEC命令を使った実行結果のダンプ・リスト



8.11

●入出力命令

入出力命令を理解するには，まず「ポート」と「ポート・アドレス」という概念を理解しておかなければなりません．5.4章で「I／Oポート」の概要を解説していますので，Figure-5.4.1と共に，もう一度目を通しておいてください．

ここでは具体的に，PC-8001，PC-8001MKII，PC-8801のブザーとプリンタのポートを例に，入出力命令の解説を行ないましょう．それらのポートの様子を次の図に示します．

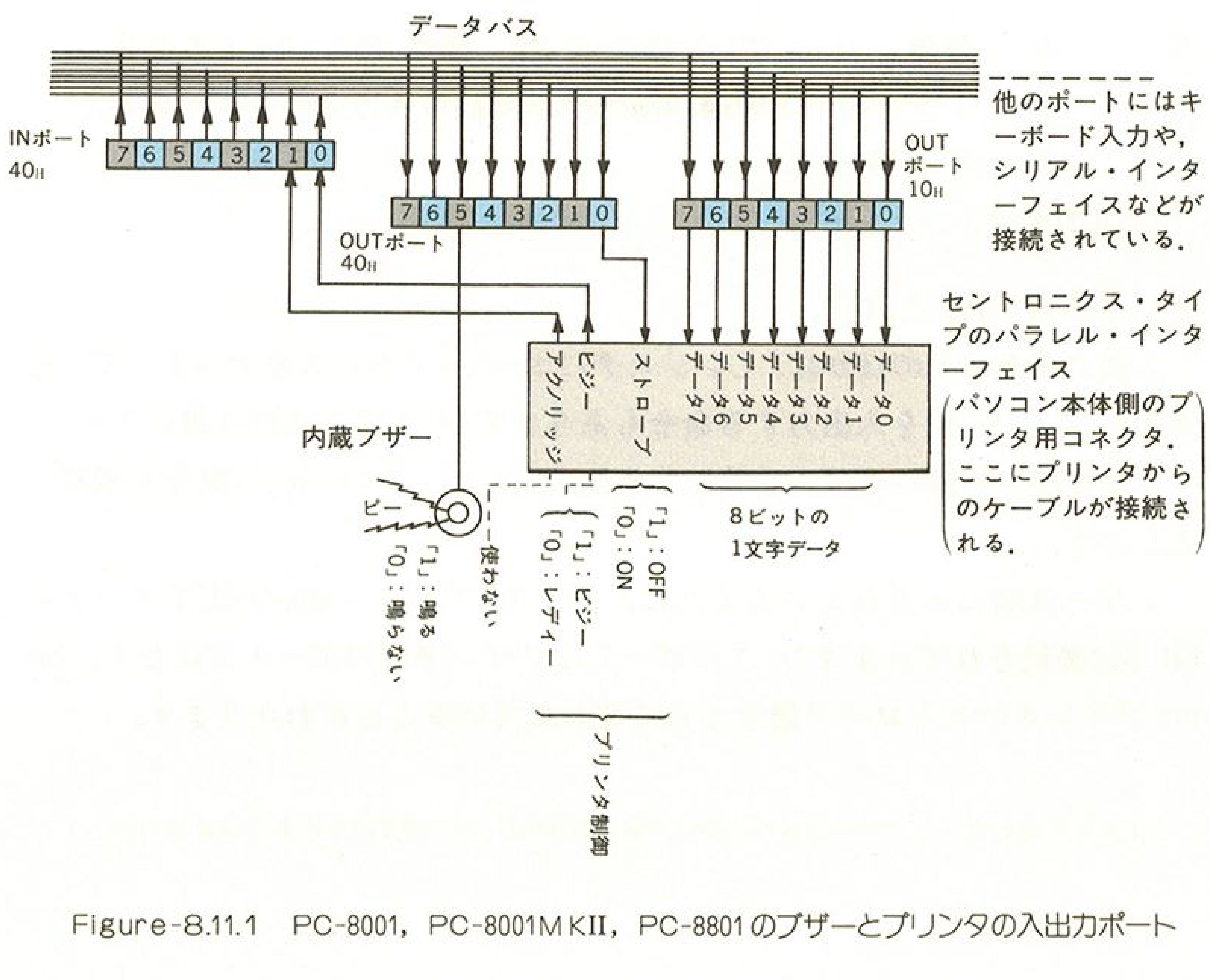


Figure-8.11.1　PC-8001，PC-8001MKII，PC-8801のブザーとプリンタの入出力ポート

コンピュータは，自分のメインメモリの操作以外は，すべて入出力ポートを介してそれに接続されているものを操作すると考えてよいでしょう．

キーボード，プリンタ，ディスク，オーディオカセット，RS-232Cインターフェイスなどの外部の装置との接続はもちろんのことで，内部的にもCRTコントローラ，DMAコントローラ，メモリバンク選択，割込みコントローラ，各種システム・コントローラなどは，「入出力ポート」を介してデータバスやアドレスバスと結ばれているのです．

では命令の解説からはいっていきましょう．

まずOUT命令は，

```
OUT    (●●)， A………マシン語では「D3  ●●」の２バイトの命令．
 ↑      ↑     ↑
 │      │     └──Aレジスタの内容を，
 │      └──ポート●●(00H～FFHの値)に，
 └──出力せよ．
```

IN命令は，

```
IN       A，(●●)………マシン語では「DB  ●●」の２バイトの命令．
 ↑       ↑    ↑
 │       │    └──ポート●●(00H～FFHの値)に入力されるデータを，
 │       └──Aレジスタに，
 └──入力せよ．
```

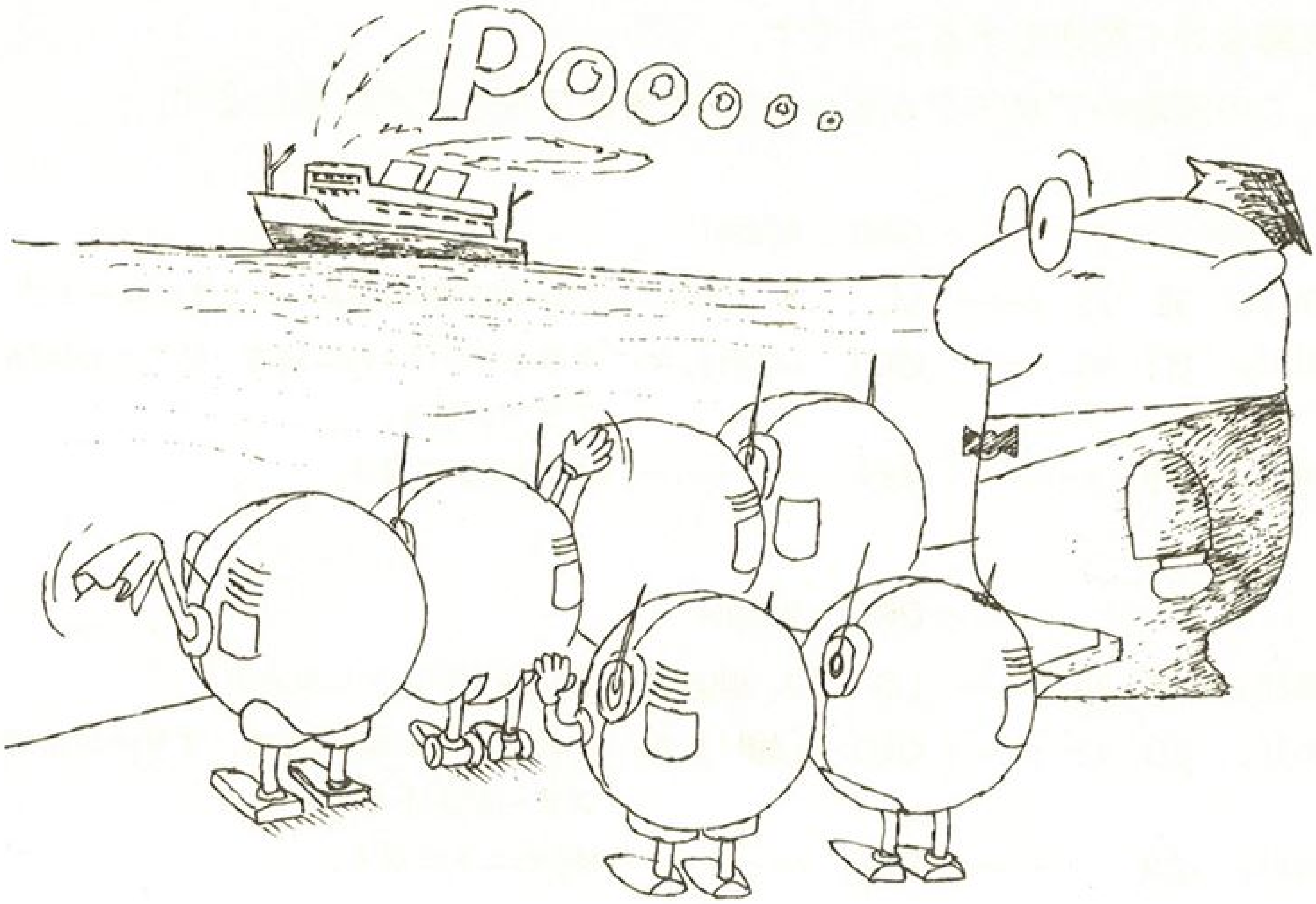
という形式で表わします．

入出力命令はそのほかに，Cレジスタにポート・アドレスをセットして，任意のレジスタの値を入出力する命令もありますが，本書では取り扱いません．

では実際に内臓のブザーを鳴らす実験を行ない，「ポート」の概念を理解しましょう．

ブザーは図に示されているように，ポート・アドレス40_HのOUTポートのbit 5に接続されています*．このポートはブザー専用のポートではなく，bit 0はプリンタのストローブ信号として使われていることがわかります．

＊ここに示してあるポートアドレスはPC-8001，PC-8001MKII，PC-8801のみにあてはまるもの．



ブザーを鳴らすには，このOUTポートのbit 5に「1」を立てればいいのです．止めるのは同じくbit 5を「0」にすればいいわけです．

では実験してみましょう．bit 5が「1」であるデータは，

$\underbrace{0010}_{2}\ \underbrace{0000}_{0}=20_H$(他にもいろいろ考えられます)

bit 5が「0」であるデータは，

$0000\ 0000=00_H$(同上)

としてOUT命令を実行してみましょう．

ただし，このOUTポート40_Hは，ブザーやプリンタ・ストローブ以外にも，コンピュータ内部の各種コントロールなどに使われていますので，単純に「0010 0000」や「0000 0000」を出力した場合，何らかの別のコントロールをしてしまう可能性もあります．ただし，PC-8001および8801では特に実験に

支障もなく動作をするようです.

この実験のプログラムをハンド・アセンブルして次に示します.

```
                   ┌------------------ ORG   9000H
                   ↓
9000H   3E 20  ←──── LD    A ,20H  …20H=00100000をAレジスタにロード.*
9002H   D3 40  ←──── OUT   (40H),A …それをポート40Hに出力. ブザーのbitをON.*
                                    ブザーが鳴る.
9004H   C9     ←──── RET   ……………MCモニタに戻る.

                   ┌------------------ ORG   9010H
                   ↓
9010H   3E 00  ←──── LD    A, 00H …00HをAレジスタにロード.*
9012H   D3 40  ←──── OUT   (40H),A …それをポート40Hに出力. ブザーのbitをOFF.*
                                    ブザーが止まる.
9014H   C9     ←──── RET   ……………MCモニタに戻る.
```

このオブジェクト・プログラムをMCモニタでメモリに書き込み, 実行してみます. アドレス9000_Hから実行するとブザーが鳴り, 9010_Hから実行すると鳴っているのが止まることが確認できます.

9000_Hから実行したとき, 鳴りっぱなしになる理由は, 普通の出力ポートは, 出力したときのデータを次のデータが出力されるまで保持する「ラッチ」と呼ばれる機能があるからです.

ここでもう一度Figure-8.11.1のブザーのところの図を見てください.「ポート」とそれに接続されている装置(この場合はただのブザー)と, 出力命令の関係がある程度理解できたことと思います.

ここではブザーを使って出力命令を実習しましたが, 入力命令も出力が入力となり, データの流れる方向が逆になるだけで, 考え方は同じです. 入力命令についての実習は, 次の章のプリンタ (Figure-8.11.1で示したもの) を動作させるプログラムの中で行ないます.

*これらのポート・アドレスや該当ビットは機種により異なります. APPENDIX 2を参照.

8.12

●その他の命令

8.1章の冒頭で，Z-80CPUの命令群は非常に豊富ですべての命令の組合せは約700にもおよぶ，と述べました．本章ではコンピュータの動作の基本となる，いくつかの重要な命令を実習しながら解説してきましたが，これまでの命令群以外にも，まだ多くの大切な命令があります．その中からいくつかをここにまとめて解説しておきます．

●● NOPとHALT命令 ●

NOP(No Operation)命令と，HALT命令は「CPUコントロール命令」と呼ばれる命令群の中の命令です．

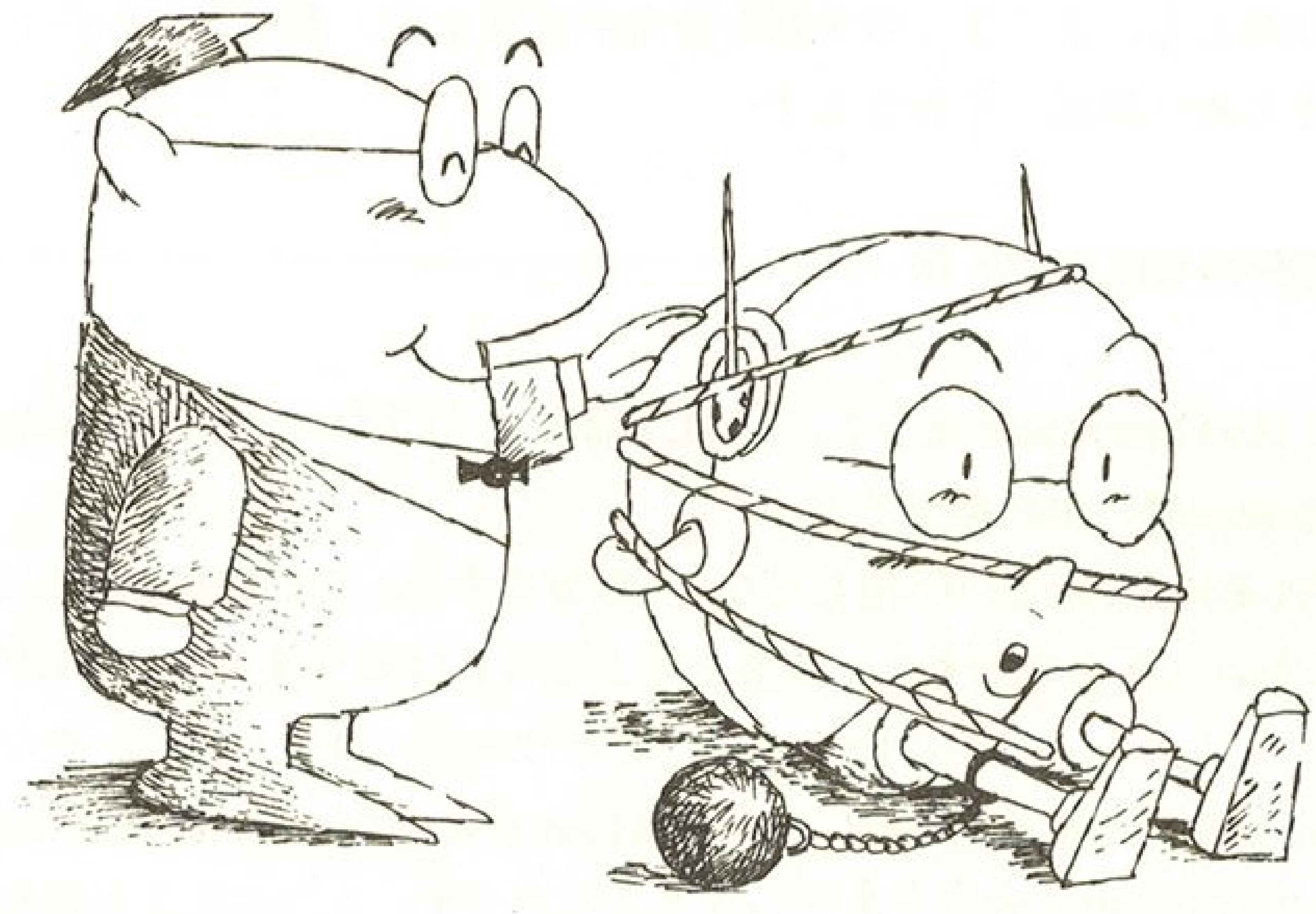
NOPはその名のとおり「何もしない」命令ですが，この認識は正しくなく，「何もしないことを実行する」命令と考えなければなりません．NOP命令も「命令」である以上，CPUに取り込まれてその命令の解析を受け，その結果が実行されます．つまり「何もしないで次の命令に進む」という命令であって，それをCPUが忠実に実行するということなのです．ここのところの考え方は非常に大切なことであり，NOP命令を実行するにも「時間がかかる」ということを知っておかなくてはなりません．NOP命令を実行するのに必要な時間は，8ビット・ロード命令の各レジスタ間でデータを転送する命令の場合と同じです(4 MHzのクロックを使った多くのマシンではおよそ1マイクロ秒)．よってNOP命令は，プログラムの実行時間を調整する「タイマー」としても

利用されます.

NOP……マシン語では「00」の1バイトの命令.

NOP命令は，ソース・プログラム上のどこのステップに挿入しようと，実行時間に関すること以外はプログラムの実行にまったく何の変化も与えません.

よって，特にハンド・アセンブルする場合などは，プログラムの一部を変更する場合を考慮して「パッチ用」としてのメモリエリアを確保する目的のために，プログラムの適当な個所にいくつかのNOP命令をあらかじめ入れておくこともあります.



次にHALT(ホルト)命令です.

HALT………マシン語では「76」の1バイトの命令.

HALT命令はCPUのプログラム・カウンタの機能を停止させる命令です.

よってこの命令を実行すると，CPUの働きは停止したままになり，どのような命令であろうと受け付けることができなくなります．CPUを再び動作状態にするには，ハードウェア的な「割込み」という操作を行なうか，CPU全体の「リセット」をかける（パーソナル・コンピュータのリセットボタンにあたる）かのいずれかふたつの方法しかありません．

HALT命令の動作確認は，一般のパーソナル・コンピュータでは，スクリーンの表示関係やその他に割込みを使っている機種が多いので困難です．HALT命令でCPUが停止しても，何らかの割込みがはいってすぐ解除されてしまうからです．

●● ブロック転送命令 ●

4.2章と6章で実行したブロック転送のプログラムは，「LDIR」という便利なブロック転送命令を使っています．8080や8085にはこの命令はありませんので，ブロック転送を行なう場合は，少々プログラムを書かなければなりません．このプログラムは次の9.1章で作成していますので参照してください．

ここでは「LDIR」と「LDDR」のふたつのブロック転送命令について解説しましょう．

LDIR ………マシン語では「ED　B0」の2バイトの命令
LDDR………マシン語では「ED　B8」の2バイトの命令．

どちらも，転送しようとするブロックの最初のアドレスをHLレジスタペアに，転送先の最初のアドレスをDEレジスタペアに，転送するバイト数をBCレジスタペアにセットします．この命令を実行することにより，自動的にブロック転送が行なわれます．上記のふたつの命令は，結果的にはまったく同一のことを実行しますが，実行過程が異なるので，場合によって次のように使い分けます．

LDIR命令は，次の図でも示すように，ブロックの上（下位アドレス）から順

に１バイトずつCPUに読み込んでは転送先のアドレスにストアし，この作業を高位アドレスに向ってくり返します．

一方，LDDR命令は，転送順が逆で，ブロックの下(高位アドレス)から順に１バイトずつCPUに読み込んでは転送先のアドレスにストアし，この作業を下位アドレスに向かってくり返します．

よって，転送するもとのブロックと，転送先のブロックが重ならなければ，どちらの命令を使っても正しく転送できますが，ブロックが重なっている場合は，次の図のように使い分けなければ正しく転送できません．使い分けを誤ると，自分のデータで自分の別の位置のデータを書き換えてしまい，もとのデータが壊されてしまうことになりますので注意が必要です．

この命令の実習についは4.2章と，６章を参照してください．

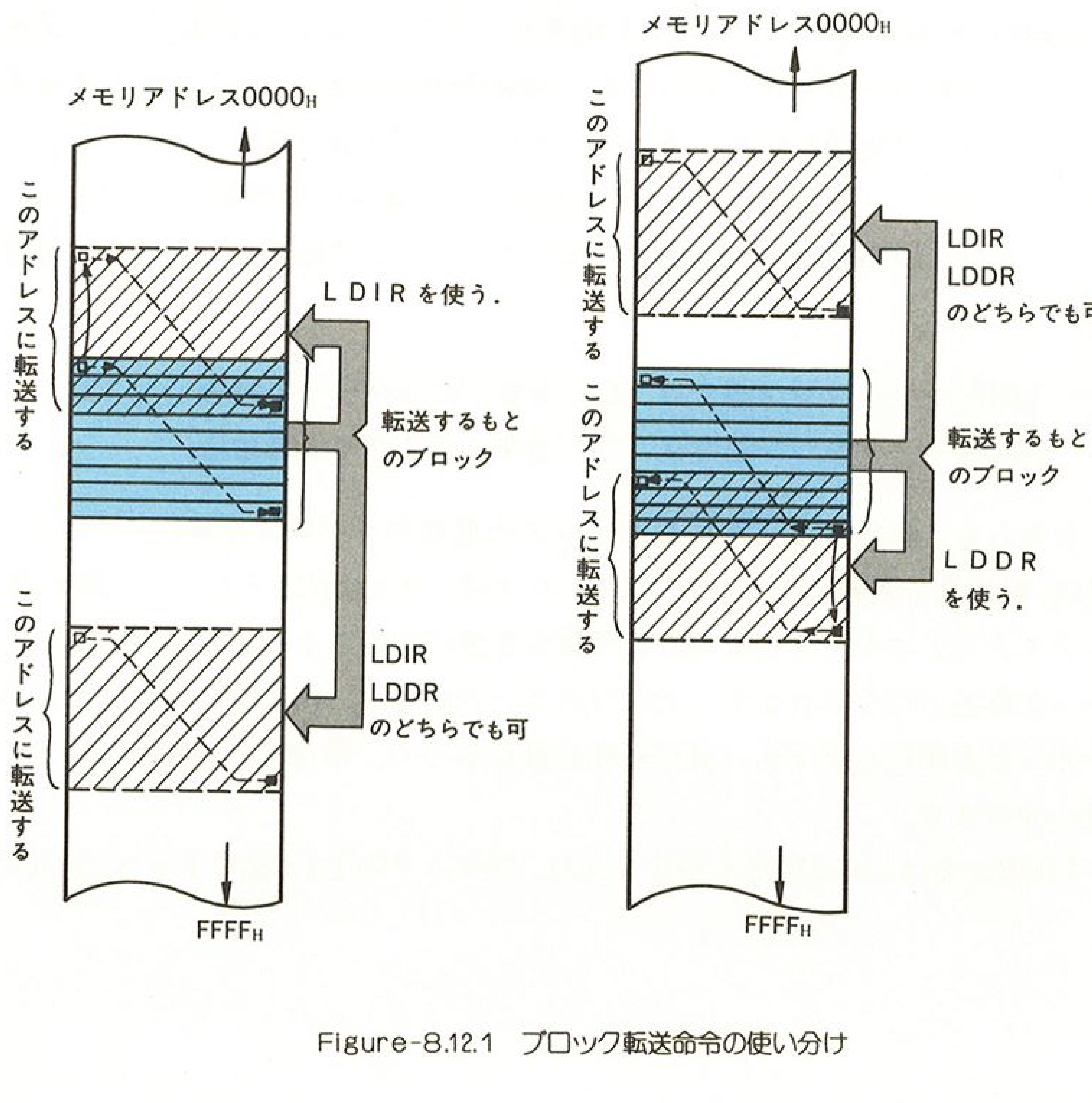


Figure-8.12.1　ブロック転送命令の使い分け

9

やさしいプログラミング例

今まで実習してきた主要な命令を組み合わせて，何か目的のあるひとつのプログラムを作成してみましょう．目的を実現するプログラムは幾通りも作り方があるものですが最初からスマートで洗練されたプログラムを組む必要はありません．原始的な手法でかまいませんから，誰が見てもわかりやすく，プログラムの流れがはっきりした作り方をするように心掛けてください．グチャグチャなプログラムは絶対にいけません．メモリを節約する必要もありません．容量の決められたROMに無理に押し込めるような場合は別ですが，ディスクやテープの場合なら苦労して少々圧縮したところでたいした得にはなりません．そんなことよりわかりやすいプログラムを書いた方がずっと良いのです．

　ではいくつかのやさしいプログラムを作ってみましょう．

9.1

●ブロック転送プログラム

4.2章，6章および8.12章でブロック転送命令(LDIR)を使った実習をしていますが，もしこの便利な命令がなかったとしたら(8080，8085にはない)どのようにしてブロック転送を行なうか考えてみましょう．

次に，6章で行なっているのとまったく同じ機能を実現したプログラムを一例として示します．

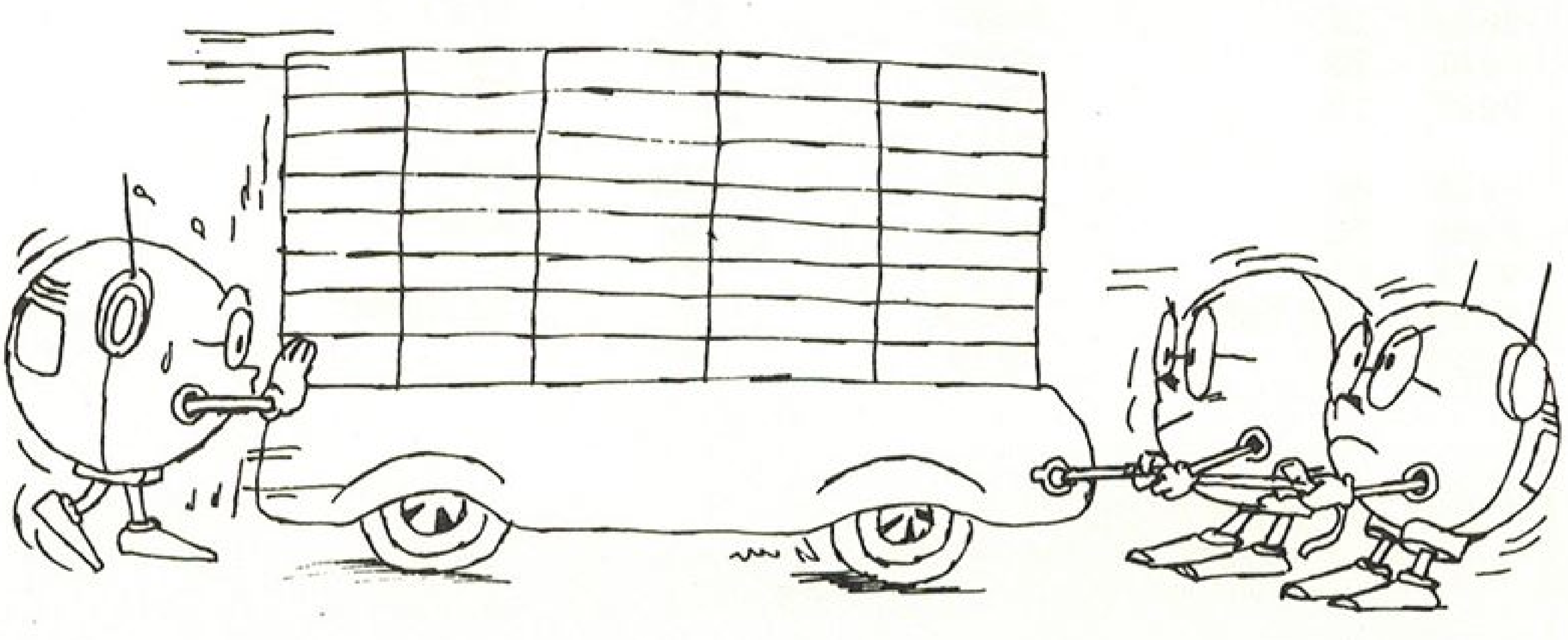
```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15
                          ラインNo.        ソース・プログラム
ロード・  オブジェクト・
アドレス  プログラム
                          0001            ORG     9000H
                          0002 ;
9000      210000          0003           {LD      HL,0000 ……………… ①
9003      1100A0          0004  初期設定 {LD      DE,0A000H …………… ②
9006      016400          0005           {LD      BC,0064H …………… ③
                          0006 LOOP:
9009      7E              0007           {LD      A,(HL) ……………… ④
900A      12              0008           {LD      (DE),A ……………… ⑤
900B      23              0009  転送実行 {INC     HL }
900C      13              0010           {INC     DE } ……………… ⑥
                          0011 ;
900D      0B              0012           {DEC     BC ………………… ⑦
900E      78              0013           {LD      A,B }
900F      B1              0014 転送バイト数{OR     C   } ……………… ⑧
9010      C20990          0015 チェック  {JP      NZ,LOOP ……………… ⑨
9013      C9              0016           {RET ……………………………… ⑩
```

①………転送するデータのTOPのアドレス値をHLにロード．
②………転送先の先頭のアドレスをDEにロード．
③………転送バイト数 64_H =100をBCにロード．
④………HLの値がアドレスするメモリの内容をAにロード．
⑤………それをDEの値がアドレスする転送先メモリにストア．
⑥………HL，DEをそれぞれ＋1．
⑦………転送するバイト数－1．
⑧………BもCも同時に0ならZフラグが立つ．100バイト終了．
⑨………BC＝00でないなら次のバイトの転送へ．
⑩………MCモニタに戻る．

Figure-9.1.1 ブロック転送命令を使わないブロック転送プログラム

このプログラムでは，転送バイト数チェックの部分で，BCレジスタペアが，BもCも同時に00_Hになるとき（つまり$BC=0000_H$のとき）をどうやって知るかが問題です．というのは，16ビット演算のINCおよびDEC命令はフラグにいっさい関係しないので，そのままフラグをみていてもダメなのです．そこでこの例ではBレジスタの内容をAレジスタに移してから，AレジスタとCレジスタ間のORをとって解決しています．本章のまえがきに反して，ずいぶんスマートな手法と言えます．

このオブジェクト・プログラムをMCモニタでメモリに書き込み，実行してみてください．4.2章で行なったのと同じ結果が得られるでしょう．



9.2

●スクリーンへのキャラクタ表示

PC-8001，8801およびPC-8001MKIIのN-BASICモードでは，Aレジスタに表示したいキャラクタのコードを持ってアドレス0035_HをCALLすることにより，スクリーンのカーソルの位置に任意の文字や記号を表示することができます*.

これは，カーソルのコントロールも含んだ「1文字スクリーン出力サブルーチン」がN-BASICのROM内に用意されており，そのルーチンの入口(エントリ・ポイントと言う)がアドレス0035_Hということなのです.

このサブルーチンを利用して，持っているキャラクタのすべてをスクリーンに表示させてみましょう.

巻末付録のキャラクタコード表をご覧ください. キャラクタコードというのは，たとえば1バイトの値「41_H」をスクリーンの表示ルーチンに送り込めば「A」という表示になり，「42_H」を送り込めば「B」という表示，「61_H」なら小文字の「a」，「$B1_H$」ならカナの「ア」，「30_H」なら数字の「0」，「20_H」なら「スペース」というように，1バイトの値(00_H～FF_H)を記号や文字，それにコントロール・キャラクタに割り当てたものです**.

キャラクタコード表から，00_H～$1F_H$と，$F8_H$～FF_Hの間は表示キャラクタではないことがわかります. よって作成するプログラムでは，20_H～$F7_H$の範囲のみのキャラクタを表示させることにします. スクリーン上の表示形式は，キャラクタコードの20_H(スペース)から順に1行に16文字づつ表示させることにしましょう.

* 他の機種ではアドレスも異なるし，キャラクタを持つレジスタがAレジスタでない場合もある. APPENDIX 2 を参照.
** Figure-2.2.4，2.2.5（Dコマンドによるアスキー表示）を参照.

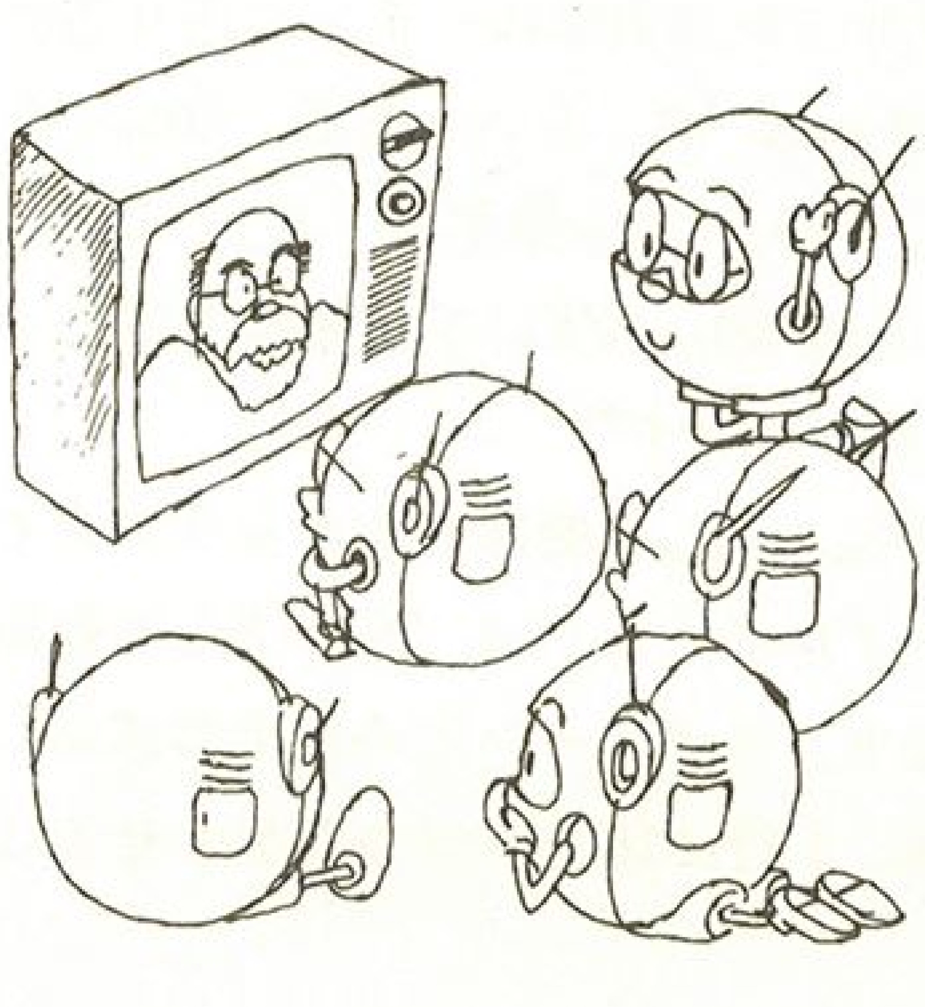
ロード・アドレス / オブジェクト・プログラム / ラインNo. / ソース・プログラム

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

                  0001            ORG     9000H
                  0002 ;
9000  3E20        0003            LD      A,20H
9002  0E10        0004            LD      C,10H
                  0005 LOOP:
9004  F5          0006            PUSH    AF
9005  C5          0007            PUSH    BC
9006  CD3500      0008            CALL    0035H*
9009  C1          0009            POP     BC
                  0010 ;
900A  0D          0011            DEC     C
900B  C21A90      0012            JP      NZ,NOCRLF
900E  3E0D        0013            LD      A,0DH
9010  CD3500      0014            CALL    0035H
9013  3E0A        0015            LD      A,0AH
9015  CD3500      0016            CALL    0035H*
9018  0E10        0017            LD      C,10H
                  0018 NOCRLF:
901A  F1          0019            POP     AF
901B  3C          0020            INC     A
901C  FEF8        0021            CP      0F8H
901E  DA0490      0022            JP      C,LOOP
9021  C9          0023            RET

Errors            0
```

Figure-9.2.1　すべてのキャラクタをスクリーンに表示するプログラム



スクリーン上の復帰，改行のコードは「$0D_H$」，「$0A_H$」であり，この2バイトを16文字表示するごとに挿入していけば，1行に16文字の表示となります．

キャラクタコード20_H～$F7_H$のキャラクタをスクリーンに表示するプログラムを上に示します．オブジェクト・プログラムをMCモニタでメモリに書き込み，9000_Hから実行してみてください．その実行結果を右に示します．

*このアドレス値は各BASICにより異なる．

……最初のキャラクタコード20Hを A レジスタにセット.
……1行を16文字とするためのカウンタを C レジスタにセット(16＝10H).

……現在のキャラクタコードを退避.
……16文字のカウンタを退避.
……1文字出力サブルーチンをコール．A レジスタのキャラクタが出力される.
……16文字のカウンタを復帰.

……カウンタを－1.
……カウンタが0にならなければ NOCRLF にジャンプ.
……カウンタが0(1行に16文字を出力済)なので復帰コードを A レジスタにセット.
……復帰改行を出力．新しい行となる.

……16文字カウンタに新しく16＝10H をセット.

……現在のキャラクタコードを復帰.
……＋1して次のキャラクタコードを A レジスタにセット.
……そのキャラクタコードと最終コード＋1(F8)とを比較.
……最終コード以前のものであれば LOOP へジャンプ.
……最終キャラクタを表示したので MC モニタに戻る.

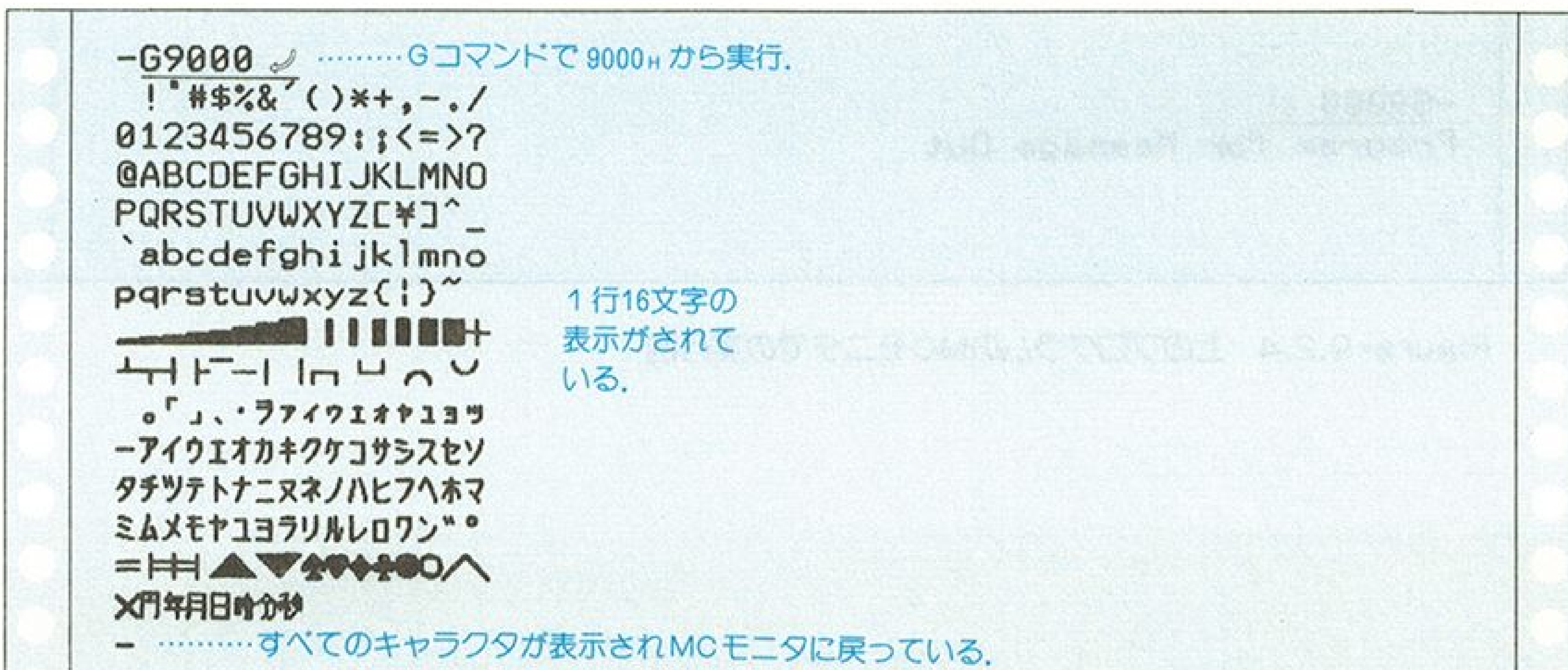


Figure-9.2.2 全キャラクタが表示された結果

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15

ロード・   オブジェクト    ラインNo.        ソース・プログラム
アドレス   ・プログラム

                          0001           ORG    9000H
                          0002 ;
9000       210F90         0003           LD     HL,MESSAGE
                          0004 LOOP:
9003       7E             0005           LD     A,(HL)
9004       B7             0006           OR     A
9005       C8             0007           RET    Z
9006       E5             0008           PUSH   HL
9007       CD3500         0009           CALL   0035H*
900A       E1             0010           POP    HL
900B       23             0011           INC    HL
900C       C30390         0012           JP     LOOP
                          0013 ;
900F       50726F67       0014 MESSAGE:  DB     'Program for
           72616D20                             Message Out',
           666F7220                             0DH,0AH,00
           4D657373
           61676520
           4F75740D
           0A00

Errors                    0
```

(注記: 50 72 6F 67 → P r o g; 4F → O; 75 74 → u t; 0A00 の 00 → メッセージの終わりを示す 00H. ／ 0DH → 復帰, 0AH → 改行, 00 → メッセージの終わりを示す 00H; 'Program for Message Out', 0DH,0AH,00 → 表示するメッセージ.**)

Figure-9.2.3 メッセージ表示のプログラム例

```
-G9000 ↵
Program for Message Out

_
```

Figure-9.2.4 上のプログラムのMCモニタでの実行例

* このアドレス値は各BASICにより異なる.

** この3行は，実際のリストでは続けて1行になっている．ここでは本の大きさの都合により，3行にわけている．

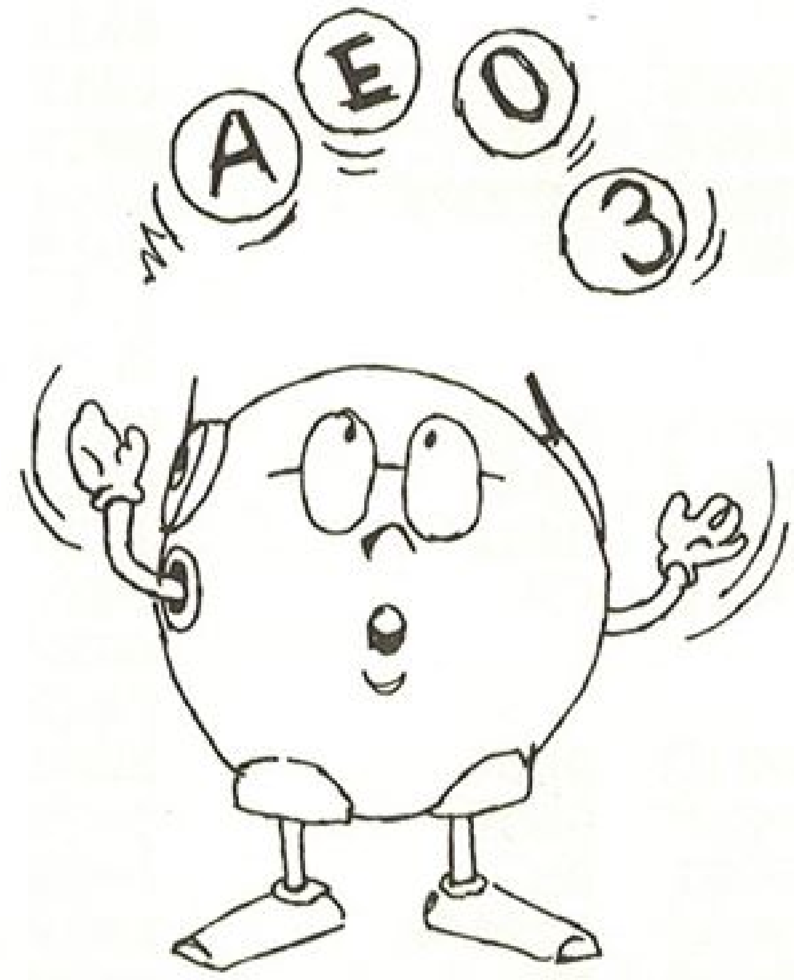
……………………メッセージエリアの先頭アドレスをHLにセット.

……………………メッセージエリアの1文字をAレジスタにロード.
……………………それは00H（メッセージの終了）か.
……………………00Hならメッセージのすべての表示終了．MCモニタに戻る.

……………………1文字出力サブルーチンのコール.

……………………HLを＋1して次の文字に進める.
……………………くり返し.

次に参考までに，任意のメッセージをスクリーンに表示するプログラムを示します．このプログラムは，表示するメッセージの最後に「00」を置き，メッセージの最初から順に，スクリーンに表示して行き，「00」が来ると終了するものです.



9.3

●ブザーの遊び

8.11章で行なった内臓のブザーを鳴らす実験にちょっと手を入れて，ピーピーピーピー……と目覚し時計の電子音のまねをしてみましょう．ブザーを鳴ら

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15
                        ラインNo.          ソース・プログラム
ロード・  オブジェクト・
アドレス  プログラム
                         0001             ORG      9000H
                         0002 ;
9000      1E20           0003             LD       E,20H
                         0004 LOOP:
9002      CD0D90         0005             CALL     BUZON
9005      CD1590         0006             CALL     BUZOFF
9008      1D             0007             DEC      E
9009      C20290         0008             JP       NZ,LOOP
900C      C9             0009             RET
                         0010 ;
                         0011 BUZON:
900D      3E20           0012             LD       A,00100000B
900F      D340           0013             OUT      (40H),A
9011      CD1D90         0014             CALL     TIMER
9014      C9             0015             RET
                         0016 ;
                         0017 BUZOFF:
9015      3E00           0018             LD       A,00000000B
9017      D340           0019             OUT      (40H),A
9019      CD1D90         0020             CALL     TIMER
901C      C9             0021             RET
                         0022 ;
                         0023 TIMER:
901D      0E50           0024             LD       C,80
901F      06B0           0025 TM1:        LD       B,0B0H
9021      05             0026 TM2:        DEC      B
9022      C22190         0027             JP       NZ,TM2
9025      0D             0028             DEC      C
9026      C21F90         0029             JP       NZ,TM1
9029      C9             0030             RET

Errors                   0
```

(C,80 の注記) この値でブザーのON，OFFの持続時間が変わる．

Figure-9.3.1 ブザー遊びのプログラム

す基礎的なことに関しては8.11章を参照してください.

ピーピーピー……と断続的に鳴らすには，ブザーがつながっているポートのブザー用のビットを適当な間隔で1→0→1→0……とすればよいことです．あまり難しくありませんね.

ではこのプログラムの一例を次に示します．このプログラムのブザーのポート・アドレスなどはPC-8001／8801のものですので，他の機種をお使いの方は注意が必要です.

このオブジェクト・プログラムをMCモニタでメモリに書き込み，実行してみてください．まるで目覚し時計という感じでピーピー鳴ります．くり返しの回数や，タイマーの値などを変えていろいろ試すとおもしろいと思います.

……ブザー音(ピーピーピー)を20_H =32回くり返すためのカウンタをEレジスタにセットする.

……ブザーを鳴らすサブルーチンをコール．ピーと鳴る.
……ブザーを止めるサブルーチンをコール．しばらく鳴り止む.
……カウンタを－1する.
……まだ0でなければLOOPにジャンプしくり返す.
……32回ピーピーピーと鳴ったら終わり，MCモニタに戻る.

……ブザーON．8.11章を参照.

……しばらくそのままの状態を保つ．タイマーのサブルーチンをコール.
……BUZONサブルーチン終わり.

……ブザーOFF．8.11章参照

……しばらくそのままの状態を保つ．タイマーのサブルーチンをコール.
……BUZOFFサブルーチン終わり.

……二重のループをまわることにより時間を費すタイマーのサブルーチン.

9.4

●プリンタを動かすプログラム

たいていのパーソナル・コンピュータは，プリンタのインターフェイスとして「セントロニクス仕様」とコンパチブルなパラレル・インターフェイスを備えています．このインターフェイスの構成は，8章のFigure-8.11.1に図示されていますので参照してください．

BASICユーザーは，LPRINTやLLISTでプリンタを使いますが，ここではプリンタを最終的に動かすマシン語のプログラムを作ってみましょう．プリンタを動作させるのは意外と簡単なのです．

プリンタを動作させる手順を次に示します．Figure-8.11.1も参照してください．

まずコンピュータはプリンタがBUSY(ビジー)かどうかをたずねます．つまり仕事中かそれとも暇なのかを調べます．具体的には図のINポート40_Hのbit 0が何かをチェックするわけです．

↓

もしプリンタがBUSY(仕事中)ならば，その仕事が終わってREADY(レディー)になるまで待ち続けます．具体的には前のチェックをくり返し行ない続けるわけです．

↓

待ち続けたかいがあってREADYになったら，コンピュータは直ちに，印字すべき8ビットのデータをプリンタに送り付けます．具体的にはOUTポート10_Hから8ビット・パラレルで印字データを出力するわけです．

↓

印字データを送り付けただけではプリンタはそのデータを受け取らないので,「データを送り付けたのでそれを取り込め」という「ストローブ」指令を別に出してやらなければなりません. 具体的にはOUTポート40_Hのbit 0からプリンターにストローブ信号を出力します.

↓

ストローブ信号を受け取ったプリンタは, 直ちに印字データを取り込み, プリンタ内部の処理にはいります. 同時にBUSY状態にして, 処理が終わらなければ次の印字データを受け付けないようにします.
この時点で印字データをため込んだり(バッファリング), 印字したりします.

↓

コンピュータは, もう必要のないストローブ信号をOFFにします.
以上でコンピュータからプリンタに1文字が引き渡されたわけです.
あとはこのくり返しになり, 復帰, 改行などのコードをプリンタが受け取れば印字することになります.

Figure-9.4.1 セントロニクス仕様・プリンタのデータ受け渡し手順

上の手順に従って, プリンタへの一文字出力ルーチンを作り, そのルーチンを使ってテスト印字のプログラムを作成してみましょう. テスト印字というのは, よくプリンタの自己診断のテスト印字に見られるような, 1行ごとに一文字ずつずらしながら全キャラクタを印字するやり方です.

そのプログラムの一例を次に示します. このプログラムのプリンタのポート・アドレスなどはPC-8001／8801のものですので, 他の機種をお使いの方は注意が必要です.

このオブジェクト・プログラムをMCモニタでメモリに書き込み実行してみてください. 次のような印字をするはずです. この印字の最終は, 右端にキャラクタコード表の最終文字であるコード「F7$_H$」が来ると当プログラムを終了します.

```
CROMEMCO CDOS Z80 ASSEMBLER CP/Mver 02.15
                         ラインNo.   ソース・プログラム
ロード・  オブジェクト・
アドレス  プログラム
                         0001            ORG    9000H
                         0002 ;
9000      CD2E90         0003            CALL   CRLFOUT
9003      0620           0004            LD     B,20H
                         0005 LOOP0:
9005      78             0006            LD     A,B
9006      0E40           0007            LD     C,40H
                         0008 LOOP1:
9008      CD1A90         0009            CALL   PRNOUT
900B      3C             0010            INC    A
900C      0D             0011            DEC    C
900D      C20890         0012            JP     NZ,LOOP1
9010      CD2E90         0013            CALL   CRLFOUT
9013      FEF8           0014            CP     0F8H
9015      C8             0015            RET    Z
9016      04             0016            INC    B
9017      C30590         0017            JP     LOOP0
                         0018 ;
                         0019 PRNOUT:
901A      F5             0020            PUSH   AF
901B      5F             0021            LD     E,A        出力キャラクタを Eレジスタに退避.
901C      DB40           0022 BUSY:      IN     A,(40H)   }
901E      E601           0023            AND    00000001B } BUSYのチェック.
9020      C21C90         0024            JP     NZ,BUSY   }
9023      7B             0025            LD     A,E       }
9024      D310           0026            OUT    (10H),A   } 出力キャラクタを出力.
9026      AF             0027            XOR    A         } ストローブをON.
9027      D340           0028            OUT    (40H),A   }
9029      3C             0029            INC    A         }
902A      D340           0030            OUT    (40H),A   } ストローブをOFF.
902C      F1             0031            POP    AF
902D      C9             0032            RET               1文字印字ルーチン終わり.
                         0033 ;
                         0034 CRLFOUT:
902E      F5             0035            PUSH   AF        }
902F      3E0D           0036            LD     A,0DH     }
9031      CD1A90         0037            CALL   PRNOUT    }
9034      3E0A           0038            LD     A,0AH     } 復帰・改行を出力
9036      CD1A90         0039            CALL   PRNOUT    } するサブルーチン.
9039      F1             0040            POP    AF        }
903A      C9             0041            RET              }

Errors                   0
```

Figure-9.4.2 プリンタを動かすプログラム

……………………復帰・改行を出力.
……………………最初のキャラクタをBレジスタにセット.

……………………1行の文字数64＝40_HをCレジスタにセット.

……………………Aレジスタのキャラクタをプリンタに出力.
……………………Aレジスタを＋1して次のキャラクタコードをセット.
……………………1行の文字数のカウンタを－1.
……………………64文字以下であればLOOP1へジャンプして次のキャラクタへ.
……………………64文字出力したので復帰・改行を出力.
……………………行の最後の文字と，最終のキャラクタコード＋1(F8)とを比較.
……………………最終のキャラクタならば当プログラムを終了.
……………………新しい行の最初のキャラクタコードをセット(旧値を＋1).
……………………新しい行の始まりへジャンプ.

……………………1文字印字サブルーチン.

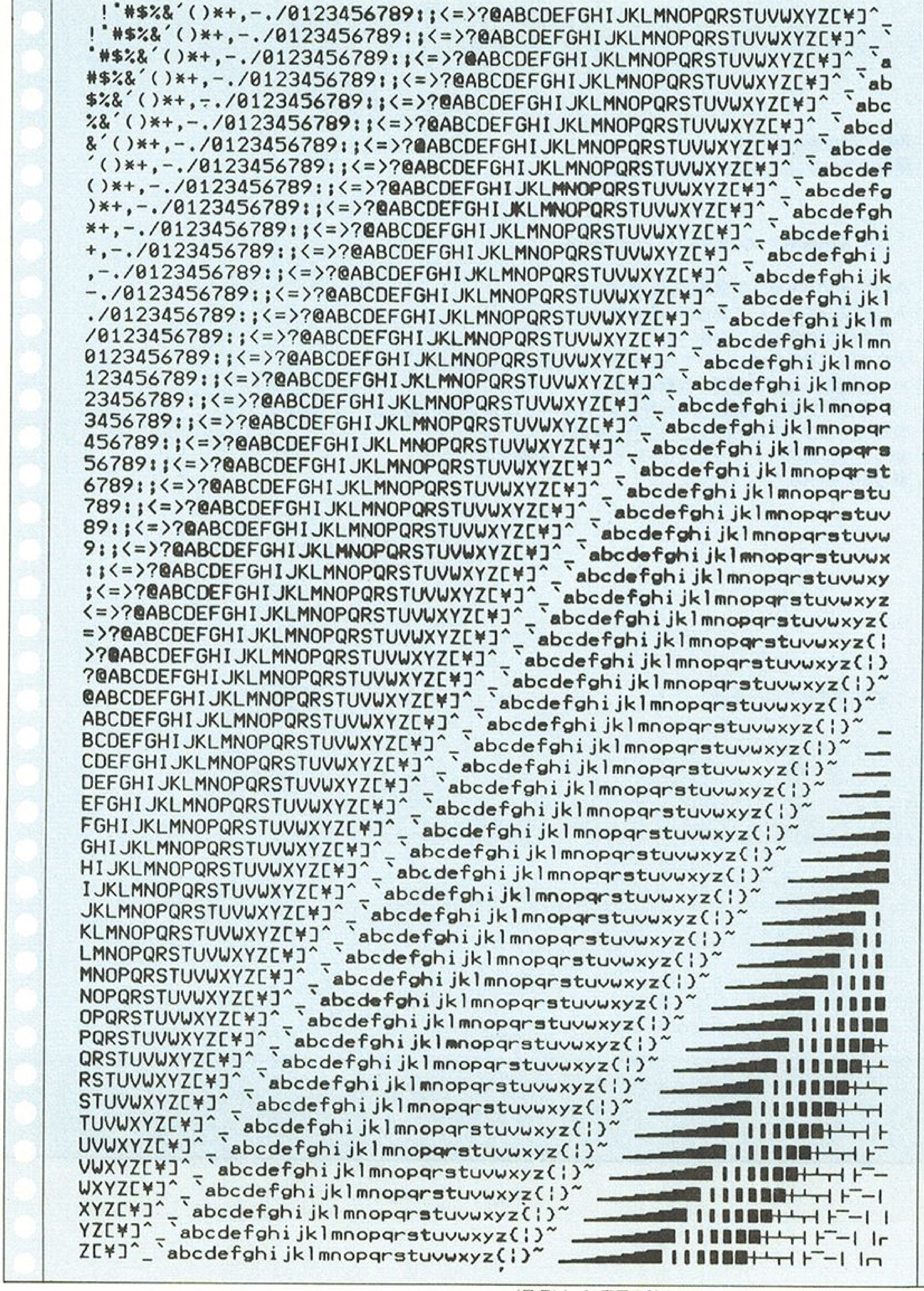


Figure-9.4.3 プログラムの実行による印字例 (スクリーン表示の時とはキャラクタコードが多少違うので注意)

終章

本書を熱心に読み進んでこられたみなさんは今，コンピュータっていったい何であったのか，その「基本的なしくみ」(本書でよく出てくる言葉である“コンピュータ・アーキテクチャ”)がひと通り理解され始めたことでしょう．そしてほかのマシン語の本も，以前よりずっとよく読めるようになっていることに気付くと思います．

パーソナル・コンピュータの電源をONにすれば即BASICマシンとして「OK」プロンプトの出てくる環境で育った人が，「コンピュータ＝BASICではなかったんだ」と目覚めるのはそう簡単なことではないでしょう．しかしその中で何人かは，良い本に出合ったり，助言を与えてくれるよい人に出合ったり，ソフトウェア環境の整ったマシンに出合ったりして，「そうか，コンピュータってBASICだけではなかったんだ」と気付くでしょう．

もしあなたがフロッピーディスクを使うようになったなら，ぜひ8ビットマシンであれば「CP／M」，16ビットマシンであれば「MS-DOS」や「CP／M-86」などのオペレーティング・システム(OS)を経験してください．さきほどの「目覚めるとき」とは，いろいろな分野のソフトウェアをOSの上で使い始めるようになったときかも知れません．フロッピーディスクを使うようになっても，いつまでもDISK BASICをやっていたのでは，コンピュータに真のパワーを発揮させることはできません．

本書の目的は，みなさんが今後，マシン語のプログラミングや，あるいは他の言語などに進もうとしている場合，それに取り掛れるだけの土台と勇気を提供するものであることは本書の冒頭で述べました．今後はあなた自身でいろんな本を読みながら進んでいかなければなりません．

ここで筆者である私が，何となく歩いてきたマイコンの世界について私事ではありますがお話ししたいと思います．みなさんの心に，今後のヒントのひとつにでもなればと思います．

私が最初に手にしたパーソナル・コンピュータ・システムは，知

る人ぞ知る往年の名機，プロセッサー・テクノロジー社のSOL-20でした．月刊ASCIIが創刊された年，1977年の秋頃の話です．この米国製のマシンは当時としてはソフトウェアもかなり完備しており，8K-BASIC，15Kの拡張BASIC，スクリーンエディタ＋アセンブラ＋デバッガのパッケージ，いくつかのテレビゲーム・ソフト等が揃っていて，カセットベースでは各＄15～＄40で入手できました．

当時の日本のパーソナル・コンピュータ・シーンはまだ胎動期にあり，1年ほど前に発売されたNECのTK-80という16進キーとLEDの表示器が8個付いただけのワンボードむき出しのマイコンの存在が目立つ程度でした．この「トレーニング・キット」と称するマイコン・キットは8080A CPUをベースに1Kバイトのモニタ ROMと1Kバイト（誤植ではなくたった1K）のRAMと，カセットのインターフェイス（これとて自分でIC等の部品を買って自作しなければならなかった）を兼ねた8255というICが乗っているだけで電源もないというシロモノでしたが，値段だけでは何と89,500円もしました．

しかしこのNECのTK-80の出現が，それ以後のパーソナル・コンピュータを含む日本の新しい文化に与えた影響は非常に大きなものがあります．とにかく日本のパーソナル・コンピュータ・シーンは，TK-80を土台として成長してきたのです．

これは私も例外ではなく，前記のSOL-20を手にする1年ほど前に，このTK-80に取り組み，その洗礼を受けた1人です．今日，日本のパーソナル・コンピュータのハードおよびソフトウェア関係の第一線で活躍をしている人の多くが，この「トレーニング・キット」に出合いそれを通過してきているようです．なぜここでTK-80の話を持ち出したのか，それをこれからお話ししましょう．

この時代にTK-80に取り組んだ人は，ある意味ではラッキーでした．その意味は，最初の最初からマシン語を相手にしなければならなかったことと，マシンがケースの中にはいっていなくてむき出しであったこと，それにマシン語の解説書を始め，応用プロ

グラム集などの各種マニュアルが，やや専門的であるけれど，割とよくできていたことなどです．TK-80にはフルキーボードやCRTディスプレイなどはありません．すべては必要最少限の機能しかない貧弱なモニタに頼って，16進キー(0～Fまでの数字と，いくつかのファンクションキー)によって，マシン語のコードを16進で打ちこむしかないのです．このことはマシン語を学習しなければ何もできないということであり，それにもめげずに喰いついていった人たちは，嫌でも，Z-80より一見難解そうな8080CPUのニーモニックに取り組まざるを得なかったのです．

またTK-80はむき出しのワンボード・マイコンキットであり，自分でハンダゴテを握り部品を一つ一つくっつけていかなければなりませんでした．つまりハードウェアに熱いおもいをして直接接触したわけです．ボードに乗っている部品は必要最少限のもので，組み立てながら，これがCPU，これがROM，これはRAM……などと，実によくコンピュータのハードウェアの基本構成がわかったものです．

しかし一方，組立てはしたものの，組立てのミスで作動しなかったり，作動したものの，アセンブラを学習しなければ何もできないことがわかり，これは自分の範疇(はんちゅう)のものではないことを悟り，投げ出してしまった人も多かったでしょう．

TK-80は正に「トレーニング・キット」そのものであり，「マイコン原理学習用の組立てキット」であったのです．

当時のこのキットの広告のうたい文句であった，**無限の可能性を秘めた**…の真の意味は，この学習用キットに無限の可能性があろうはずはなく，実はこのキットに正面から取り組んだ人が，その後いろいろな可能性を秘めることになるかも知れない，という人間的な意味であったことが最近わかってきました．このキャッチフレーズを考えたコピーライターが，そこまで考えていたかどうかは知りませんが，こう解釈してみると実に素晴らしいキャッチフレーズに思えます．もっとも当時は「フザケルナ」という感

じでしたが…….

というようなことで，私たちのマイコンとの出合いの時代，1976〜1977年はまずハンダゴテを握りハードウェアを手掛け，マシン語を手掛けなければ何も遊べなかったのです．

現在のような，近所の電器店やスーパーでコンピュータ・システムが赤札を付けて売られているシーンと，たいへんな違いだったわけです．

しかしコンピュータが除々にわかってくると，TK-80の限界がはっきり見えてきました．TK-80は実に有効な「トレーニング・マシン」でありますが，「コンピュータ・システム」ではなかったのです．コンピュータ「システム」とは，入力用のフルキーボード，出力用のスクリーン，それにシステム全体の動きをコントロールできる各種の機能を持った使いやすいモニタなどが完備され，それが有効に使えるものでなくてはなりません．

コンピュータはマシン語の一つ一つをいじり回していることではない．言語や，開発や，応用ソフトウェアをもっと自由に使えるマシンでなければコンピュータの世界は開けていかない．と思い始めた当時，まだ，日本にはパーソナル・コンピュータの姿はどこにもありませんでした(工業用のマイクロ・コンピュータ開発システムは数社から発売されていましたが)．そこで本章冒頭のSOL-20をメーカーから直接輸入することになってしまったのです．冒頭の「パーソナル・コンピュータ・システム」の“システム”の意味はこういう意味が込められているのです．

著者が使用していたSOL-20



このSOL上でアセンブラをやり，BASICをやり，スーパースタートレックのゲームをやり，しばらくは満足して遊んでいました．

当時の日本のマイコン・ソフトウェアシーンはと言うと，これも日本のコンピュータ史上画期的な出来事であるASCII誌創刊第2号

1977年8月号から始まるパロアルト版TINY BASICの全リストの紹介がありました．その時私はまだSOLを手に入れてなかったので，2，3カ月続いたこれらの記事とリストを首っ引きでASR-33(ガチャガチャとひどい騒音を発するノロいテレタイプ)をつないだTK-80に移植を試みました．数ヵ月を費やして，どうやらまともにTINY BASICが動き始めたのと，SOLが税関に到着したのはほぼ同じ時期であったように思います．

今までミニコンのアセンブラの経験はありましたが，わずか2KバイトのTINY BASICと言えども，会話形言語が動き出した時の喜びと興奮は言葉には表せません．

TK-80用のTINY BASIC 文法解説記事（月刊アスキー77年11月号)


合理的な文法規則により，直観的にプログラムが組める言語

TK-80+2KBASIC+文法解説

飯田陽一

カセットインターフェース

BASIC用 TK-80システム

このASCII誌によって具体的に私たちに紹介されたTINY BASICは，その後TINY BASICブームとでも言うべく広く普及していき，現在もその時の興奮を伝える名残りとして，その中身は変わってもASCII誌には，TINY BASIC NEWSLETTERのコーナーが続いているのです．

1977年の秋，TK-80でTINY BASICが起動し始めた頃，すでに私の机の上のSOLではスクリーンエディタ付きの8K-BASICがCRCチェック付き(チェックサム方式より高度なエラー検出方式)の1200ボーのカセットベースで働いていました．このBASICも，半年後にリリースされた15K拡張BASICも，今は使うこともなく，机の引出しの奥にしまい込まれていますが，とても良くできたBASICで，マイクロソフト社BASICとは，一味違った雰囲気を持っていました．またアセンブラなどのマシン語ソフト開発ツールもなかなかのもので，デバッガには1ステップごとのトレース実行（シュミレート実行：1ステップ実行するたびの，CPUの各レジスタの値が表示されるもの)の機能もありました．

SOLとともに，カセットベースの一応のソフトウェアをもと

に，しばらくは遊んでいましたが，このあと私にとってコンピュータに対する大きな転期があったのです．

それはフロッピーディスクとCP/MというDOS(ディスク・オペレーティング・システム)との出合いでした．

当時の米国のマイコン雑誌を秋葉原の本屋で買い求めては何となく読んでいるうちに，よくはわからないけれど「CP/M」というフロッピーディスク上で使うOSの存在に何か重大な意味があるのではないかという漠然とした意識が芽生えていました．

この時期に幸運にも，しばらく米国で遊学して帰って来たばかりの友人から，CP/Mが実動するところを見せられたのです．8インチのディスクドライブはPerSci社の270型高速シーク・ドライブ，CP/MのVersionは1.4でした．彼はこれら一式をかついで帰ってきたのです．

これだ！

ああ，コンピュータってこれだったのか．

コンピュータのハードウェアと，各種のソフトウェア，それの接点にあるディスクドライブ，そしてそれらを有機的に結びつけ，コンピュータに生命を与えるCP/Mという*OSの存在*．

私のSOLに，安く仕入れた8インチのフロッピーディスクが2台付き，CP/Mが完動したのはそれから2，3ヵ月後のことでした．もちろん彼を始め回りの人の手助けがあったからこそ可能でした．その頃の私の記事，ASCII誌1979年5月号～12月号の「How To CP/M」は，当時CP/Mに挑戦していた時の思い出です．

以上，私事を本書に持ち込んでしまいましたが，これをお読みになって，あなたは今までのパーソナル・コンピュータに対するイメージが少しは変わったでしょうか．一昔前とは格段に使いやすくなったパーソナル・コンピュータ．電源を入れれば何もしないで24Kとか32KのすばらしいBASICが起動する環境に置かれている現代のユーザー．あまりにも便利なためにBASICから出られないでいるのは当然のことかも知れません．しかしそういうあなたも，

きっとある日，

ああ，コンピュータってこれだったのか！

と思う日がきっと来るでしょう．また私にも，いつか次の新しい衝撃が来ることでしょう．

BASICを出発点としている現在の多くのユーザーにとって，コンピュータの世界はその外側に大きく広がっているのです．

本書の存在が，BASICの外側を知るための足掛りとなれば，私にとってこんなにうれしいことはありません．特に本章を読んでいただいたことをうれしく思います．どうもありがとう．これからもがんばってください．

APPENDIX

1．各機種でMCモニタを利用する場合の注意点

本書ではマシン語を学んでいく道具として，BASICのプログラムによりMCモニタを作成し，これを使って実習解説を行なっています．本文で示されている実習例などの各種リストは，NECのPC-8001，PC-8001 MK II，PC-8801のN-BASICによるものです．しかし，機種によりBASICの機能やフリーエリア，ワーキングエリアの位置（アドレス）などが異なりますので，N-BASIC以外を使う場合は次の点に注意してください．

- 機種によりMCモニタのプログラムを変更しなければならないものがある．
- 機種によりフリーエリアのアドレスが異なるため，本書で実習するマシン語プログラムをメモリに書き込む際に，アドレスなどを変更しなければならないものがある．

では，これらの注意点を各機種に組み込まれているBASICの種類ごとに示します．

1.1 N-BASIC，N80-BASIC，N88-BASIC，T-BASICの場合

これらのBASICはN-BASICと基本的な機能は共通ですので，15ページに示したMCモニタのプログラムの1000行のCLEAR文を変えるだけですみます。実習でマシン語のプログラムを書き込むアドレス（フリーエリア）については，機種ごとに若干異なります．次に各BASICについてのCLEAR文のパラメータ，フリーエリアを示しますので参考にしてください．

BASIC の種類		CLEAR 文	フリーエリア
N-BASIC	ディスクなし	300, &H8FFF	9000H-E9FFH
	ディスク使用	300, &H8FFF	9000H-D025H
	RAM が 16 K (旧 PC-8001 のみ)	300, &HCFFF	D000H-E9FFH
N_{80}-BASIC	ディスクなし	300, &H8FFF	9000H-E5FFH
	ディスク使用	300, &H8FFF	9000H-C43FH
N_{88}-BASIC	ディスクなし	, &H8FFF	9000H-E5FFH
	ディスク使用	, &HC3FF	C400H-E5FFH
T-BASIC	ディスクなし	, &H8FFF	9000H-F7FFH
	ディスク使用	, &HA7FF	A800H-FEEFH

● ディスク使用時の値は，ファイルバッファをふたつとったときのものです．

● N_{80}-BASIC, N_{88}-BASIC を使ってもかまいませんが，N-BASIC モードで使用すれば本書どおりの実習ができるので便利です．

■フリーエリアを9000_Hから使用できない場合の注意

先の表でわかるように，RAM が 16 K バイトの N-BASIC（旧 PC-8001 のみ）とディスクを使用した N_{88}-BASIC およびT-BASICの場合は，本文の例題や実習などのマシン語プログラムで使用する9000_H台のメモリ・アドレスが使えません．この場合は 9000_H台の代わりに$D000_H$台を使用してください．プログラムや実行例のアドレスに関する部分の $9\times\times\times_H$を $D\times\times\times_H$に変更するわけです．実習 1 ～ 4 では変更すべき箇所に印を付けてあります．参考にしてください．

1.2 N_{60}-BASICの場合

N_{60}-BASIC は N-BASIC と異なる点があるので，MC モニタのプログラムを変更する必要があります．N_{60}-BASIC 用の MC モニタのプログラムを以下に示します．このプログラムは RAM が 16 K でも 32 K でも，また拡張 BASIC でもまったく変更なしで動きます．ただし，電源を入れたときやリセットしたとき「How many pages?」ときいてきますので，このときは 1 を入力してください．また，拡張 BASIC では「How many files?」ときいてきますが，このときは 0 を入力してください。

この MC モニタでマシン語プログラムを書き込めるアドレス(フリーエリア)は，$D000_H$から $F7FF_H$です．「フリーエリアを 9000_Hから使用できない場合の注意」を参照してください．

```
1000 CLEAR 300,&HCFFF
1010 DIM HT$(15)
1020 FOR I=0 TO 15
1030 READ HT$(I)
1040 NEXT
1050 DATA 0,1,2,3,4,5,6,7
1060 DATA 8,9,A,B,C,D,E,F
1070 CONSOLE 0,25 : CLS
1080 PRINT "        MACHINE";
1090 PRINT " CODE MONITOR"
1100 :
1110 :
1120 PRINT
1130 BU$="" : INPUT "-";BU$
1140 CM$=LEFT$(BU$,1)
1150 IF CM$="D" THEN 1230
1160 IF CM$="d" THEN 1230
1170 IF CM$="S" THEN 1640
1180 IF CM$="s" THEN 1640
1190 IF CM$="G" THEN 1830
1200 IF CM$="g" THEN 1830
1210 PRINT " What ?"
1220 GOTO 1100
1230 :
1240 :
1250 REM DUMP : MEMORY DUMP
1260 IF LEN(BU$)>1 THEN 1290
1270 ME=MS+63
1280 GOTO 1510
1290 :
1300 J=0
1310 T=LEN(BU$)
1320 FOR I=2 TO T
1330 T$=MID$(BU$,I,1)
1340 IF T$="." THEN J=I : I=T
1350 NEXT
1360 :
1370 IF J=0 THEN 1480
1380 IF J-4>2 THEN K=J-4
1390 IF J-4=<2 THEN K=2
1400 L=LEN(BU$)-J
1410 IF L>4 THEN L=4
1420 T$=MID$(BU$,K,J-K)
1430 MS=VAL("&h"+T$)
1440 IF MS<0 THEN MS=MS+65536
1450 ME=VAL("&h"+RIGHT$(BU$,L))
1460 IF ME<0 THEN ME=ME+65536
1470 GOTO 1510
1480 :
1490 GOSUB 1910
1500 MS=MM : ME=MS+63
1510 :
1520 FOR I=MS TO ME STEP 8
1530 X=I : FI=4 : GOSUB 2010
1540 PRINT X$;"  ";
1550 FOR Q=0 TO 7
1560 X=I+Q : GOSUB 1980
1570 PRINT X$;" ";
1580 IF I+Q=ME THEN Q=7
1590 NEXT
1600 PRINT
1610 NEXT
1620 MS=ME+1
1630 GOTO 1100
1640 :
1650 :
1660 REM S : MEMORY CHANGE
1670 IF LEN(BU$)=1 THEN 1690
1680 GOSUB 1910: SS=MM
1690 :
1700 X=SS : FI=4 : GOSUB 2010
1710 PRINT X$;" ";
1720 X=SS : GOSUB 1980
1730 PRINT X$;" ";
1740 ID$="" : INPUT ID$
1750 IF LEN(ID$)=0 THEN 1810
1760 T$=RIGHT$(ID$,1)
1770 IF T$<>"." THEN 1790
1780 SS=SS+1 : GOTO 1100
1790 T$=RIGHT$(ID$,2)
1800 POKE SS,VAL("&h"+T$)
1810 SS=SS+1
1820 GOTO 1690
1830 :
1840 :
1850 REM GO : EXECUTE PROGRAM
1860 IF LEN(BU$)>1 THEN 1880
1870 PRINT "What ?" : GOTO 1100
1880 GOSUB 1910
1890 EXEC MM
1900 GOTO 1100
1910 :
1920 :
1930 M=LEN(BU$)-1
1940 IF M>4 THEN M=4
1950 MM=VAL("&h"+RIGHT$(BU$,M))
1960 IF MM<0 THEN MM=MM+65536
1970 RETURN
1980 :
1990 X=PEEK(X)
2000 FI=2
2010 :
2020 X$=""
2030 FOR H=1 TO FI
2040 T=INT(X/16)
2050 X$=HT$(X-T*16)+X$
2060 X=T
2070 NEXT
2080 RETURN
```

Figure-A.1.2　N60-BASICのMCモニタのプログラム

■プログラムの解説

このモニタのコマンドの与え方は，本文のものとほぼ同じです．最初にコマンド名（d, s, gのうちのいずれか）を入力し，必要ならばそのあとに続けてアドレスを入力します．さらに必要な場合は，スペースをひとつあけて，もうひとつアドレスを入力します．例えば「d800 1000」，「s8200」，「g3」などです．

以後の解説では最初の文字を「コマンド名」，その次にあるアドレスを「第1パラメータ」，その次にあるアドレスを「第2パラメータ」と呼びます．

1000行〜1090行

初期設定を行ないます．

1010行〜1060行では，1990行からのサブルーチンで使うデータを設定しています．

1100行〜1220行

入力されたコマンドに従って，それぞれの処理ルーチンへ飛びます．BU$には，コマンド待ちのときに入力された文字列が入ります．コマンド名がおかしいときは「What?」と表示します．

1230行〜1630行

Dコマンド（メモリの内容を表示する）の処理を行ないます．MSには表示する最初のアドレス，MEには表示する最後のアドレスが入ります．1260行〜1500行でコマンド待ちのとき入力された文字列を見てMS, MEを設定し，1510行〜1630行でMSのアドレスからMEのアドレスまでの内容を表示します．

1270〜1280行は，パラメータがないときにMS, MEを設定するところです．MEには，64バイト表示されるように，MS+63が設定されます．

1290行〜1350行は，コマンド待ちのときに入力された文字列のなかからスペース（第1パラメータと第2パラメータの区切り記号）を探して，それが何文字目にあるかをJに入れます．スペースのない場合は0が入ります．N-BASICの「J=INSTR（BU$, " "）」と同じ処理です．

1380行〜1470行は，第1，第2パラメータの両方が指定されているときにMS, MEを設定するところです．このプログラムではパラメータとして1〜4文字指定したときはそのまま解釈され，5文字以上のときは右の4文字が有効になります．Kにはひとつ目のアドレス（MS）としてBU$の何文字目からとるかが，Lにはふたつ目のアドレス（ME）として何文字とるかが入ります．

1440行と1460行は，16進数のVALをると&H8000〜&HFFFFが10進の−32768〜−1に変換されてしまうために必要となります．これはVALという関数が16進数を「符号付き2進数」（本書では説明していない）とみなすために起こることですが，ここでは不都

合です．そこで 8000_H～$FFFF_H$が 32768～65535 となるように 1440 行と 1460 行があるのです．

1480 行～1500 行は，第 1 パラメータだけが与えられたときに MS, ME を設定するところです．

1510 行～1630 行では，MS のアドレスから ME のアドレスまでのメモリ内容を表示します．1620 行では，次回に D コマンドが指定されたときにパラメータがない場合に備え，コマンド待ちに戻る直前に，MS に ME＋1 を設定しています．

1640 行～1820 行

S コマンド（メモリの内容を変更する）の表示を行ないます．SS には書き換えるメモリのアドレスが入ります．このルーチンでは最後に抜け出すときに操作したアドレスの次のアドレスが SS に設定されるので，パラメータのないときは SS の設定を省略します．

1740 行では書き換えるべきデータを入力しています．ここで入力された文字列の右端がピリオドだった場合は，S コマンドから抜け出てコマンド待ちに戻ります．何も入力されずに RETURN キーが押されたときは書き換えず，何か入力されたときはその値にメモリの内容を書き換えて次のアドレスに進みます．

1830 行～1900 行

G コマンド（マシン語プログラムの実行）の処理を行ないます．

1910 行～1970 行

アドレスがひとつ指定されたときに，そのアドレスを MM に設定するサブルーチンです．

1980 行～2080 行

X にアドレスを設定してコールすると，そのアドレスの内容の 16 進 2 桁表現を X$に設定するサブルーチンです．

2010 行～2080 行

X に数を，FI に桁数を設定してコールすると，X の 16 進 FI 桁表現を X$に設定するサブルーチンです．このルーチンの機能は，N-BASIC で「X$＝RIGHT$（"000"＋HEX$（X），FI）」としたものと同じです．

■N_{60}-BASIC用MCモニタの使い方

N_{60}-BASIC での MC モニタの使い方は，基本的には本文のものと同じです．違うところは，MC モニタが入力を求める際に「？」を表示する点と，アスキー表示を行なう機能がない点です．

1.3 その他の機種の場合

本文のFigure-1.1.1のMCモニタプログラムを参考にして，各自のマンシ用のプログラムを作成してください．もしこのプログラムで動作しない場合は，N₆₀-BASICのプログラムも参考にしてください．このプログラムは割合と基本的な機能しか使っていないので，多くのマシンで利用することが可能だと思います．もしうまくいかない場合は，各マシンに独自のモニタが用意されていればそれを使うのも良いでしょう．

なお，マシン語を打ち込むフリーエリアは，機種によって異なります．9000$_H$台を使用できない機種の場合は，当APPENDIX 1.1の「フリーエリアを9000$_H$から使用できない場合の注意」を参照してください．

2．本書で使ったROM内ルーチンとポートの機種別アドレス

8章，9章で扱ったプログラムのなかには，画面に文字を出力するためのROM内サブルーチンや，直接ブザーやプリンタ用のポートを操作しているものがあります．一般に，機種によってROMの内容やポートのアドレスは異なりますから，PC-8001，PC-8001MKII，PC-8801のN-BASIC以外でこういったプログラムを実行するには，ROM内のルーチンのコールのしかたや入出力ポートのアドレスを変更しなければなりません．

以下に機種別のROM内ルーチンのコールのしかた（エントリ・アドレスと，パラメータの渡し方）とポートのアドレスを示します．ここに挙げてない機種については，各自調べてください．

なお，これによって変更すべき箇所は本文のリストに明示されています．

■画面1 文字出力の機種別ROM内ルーチン

機種	アドレス	パラメタ受渡し
PC 8801 N$_{88}$-BASIC	3E0D$_H$	Aレジスタ←キャラクタコード
PC-8001 MK II N$_{80}$ BASIC	0257$_H$	Aレジスタ←キャラクタコード
PC-6001 N$_{60}$-BASIC	1075$_H$	Aレジスタ←キャラクタコード
PASOPIA	0892$_H$	Aレジスタ←キャラクタコード
X-1	0013$_H$	Aレジスタ←キャラクタコード

■プリンタへの文字出力およびBEEP音のポートアドレスに関する注意

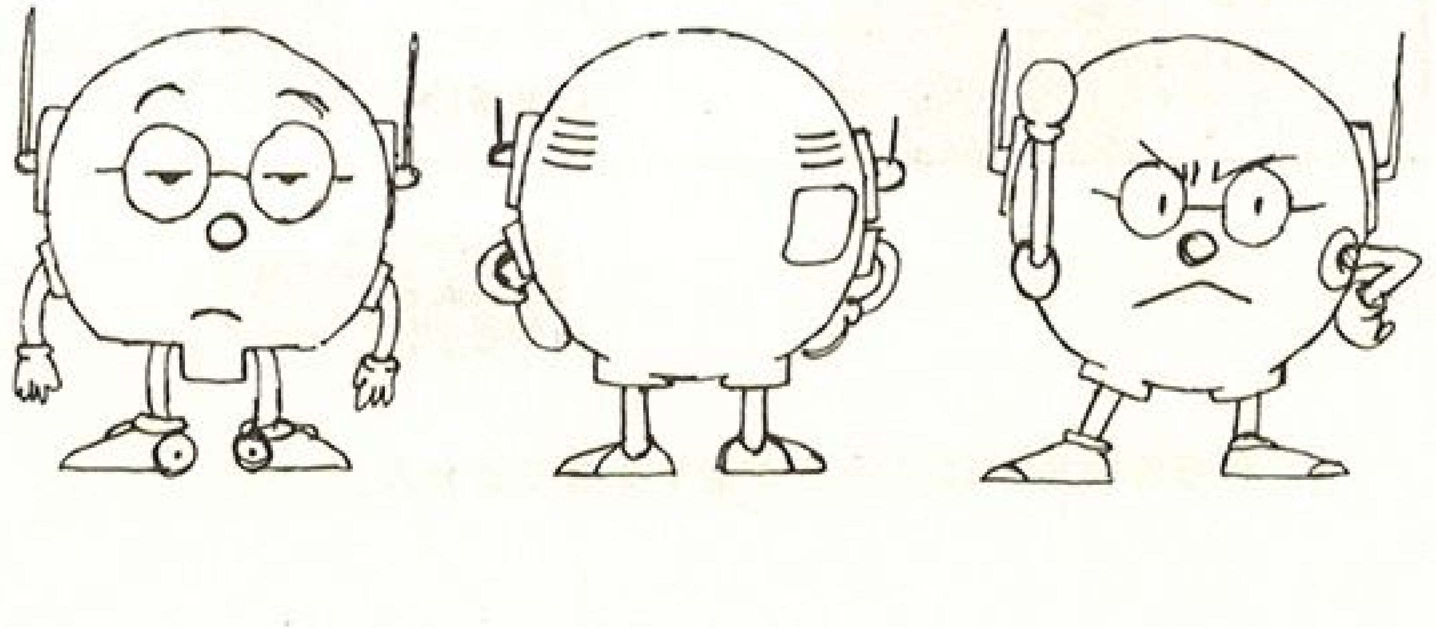
PC-8001，PC-8001MKII，PC-8801については，プリンタ用のポートアドレスやBEEP音のポートアドレスが共通していますので，本書の実習をそのまま実行することができます．

ただし，他の機種では単にポートのアドレスや各種信号などのオン，オフが違うだけではありません．機種ごとに細部の仕様がいろいろと違う場合が多いのです．たとえば，BEEP音を出すのにPC-6001やPASOPIAでは内部のシンセサイザを使用しているので，本書と同じ実習はできません．プリンタ制御についても細部の仕様が違うので，本書のプログラムのロジックをそのまま使うわけにはいかないことが多いのです．

BEEP音を出す実習やプリンタへの文字出力の実習を行なう場合は，各自でそれぞれの機種に関する情報を入手し，いろいろと工夫してみる必要があります．

CPUがZ-80の主な8 bitパーソナル・コンピュータ

NEC	PC-8001，PC-8001MKII，PC-6001，PC-6001MKII，PC-8801
SHARP	MZ-80K，MZ-80C，MZ-80K2E，MZ-1200，MZ-2000，MZ-700，MZ-2200，X1
沖電気	IF-800 model 10，IF-800 model 20，IF-800 model 30
東芝	PASOPIA，PASOPIA 7，PASOPIA 5
SONY	SMC-70
三菱電機	MULTI-8
CASIO	FP-1000，FP-1100，FP-5700
EPSON	QC-10



CPU が Z-80 でない主な 8 bit パーソナル・コンピュータ

富士通	FM-7, FM-8, FM-11
日立	ベーシックマスターレベルⅠ, レベルⅡ, レベルⅢ, レベルⅢ mark Ⅱ, レベルⅢ mark 5
松下電器	JR-100, JR-200
APPLE	APPLE, APPLE Ⅱ, APPLE Ⅱe, APPLE Ⅲ

3. 機能別主要命令表

3.1 主要8ビットロード命令

		1バイト（8ビット）のデータ	それぞれのレジスタの内容							それぞれのレジスタペアでアドレスされるメモリの内容			2バイト(16ビット)の値によってアドレスされるメモリの内容
	命令＼X	●●	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(○○●●)
Xをそれぞれのレジスタにロードする.	LD A, X	3E ●●	7F	78	79	7A	7B	7C	7D	7E	0A	1A	3A●●○○
	LD B, X	06 ●●	47	40	41	42	43	44	45	46			
	LD C, X	0E ●●	4F	48	49	4A	4B	4C	4D	4E			
	LD D, X	16 ●●	57	50	51	52	53	54	55	56			
	LD E, X	1E ●●	5F	58	59	5A	5B	5C	5D	5E			
	LD H, X	26 ●●	67	60	61	62	63	64	65	66			
	LD L, X	2E ●●	6F	68	69	6A	6B	6C	6D	6E			
Xをそれぞれのレジスタペアがアドレスするメモリにストアする.	LD (HL), X	36 ●●	77	70	71	72	73	74	75				
	LD (BC), X		02										
	LD (DE), X		12										
2バイト(16ビット)の値によってアドレスされるメモリにストアする.	LD (○○●●), X		32 ●●○○										

LD A,(1234H)
（2バイト（16ビット）の値，この例ではアドレス1234Hのメモリ内容をAレジスタにロードする.）

LD A,(HL)
（HLレジスタペアによってアドレスされるメモリの内容をAレジスタにロードする.）

LD C,E
（Eレジスタの内容をCレジスタに転送する.）

LD (HL),E
（Eレジスタの内容をHLレジスタペアによってアドレスされるメモリにストアする.）

LD L,12H
（1バイト（8ビット）のデータ，この例では12_HをLレジスタにロードする.）

LD (1234H),A
（Aレジスタの内容をこの例ではアドレス1234_Hのメモリにストアする.）

これらの命令ではフラグは全く変わりません.

3.2 主要8ビット算術論理演算命令

		命令 ＼ X	A	B	C	D	E	H	L	(HL)	●●
			それぞれのレジスタの内容							HLレジスタペアでアドレスされるメモリの内容	1バイト（8ビット）のデータ
演算結果はAレジスタにセットされる	Aレジスタの値とXを加算する.	ADD A, X	87	80	81	82	83	84	85	86	C6●●
	Aレジスタの値とXとCYフラグの値を加算する.	ADC A, X	8F	88	89	8A	8B	8C	8D	8E	CE●●
	Aレジスタの値からXを減算する.	SUB X	97	90	91	92	93	94	95	96	D6●●
	Aレジスタの値からXとCYフラグの値を減算する.	SBC A, X	9F	98	99	9A	9B	9C	9D	9E	DE●●
	Aレジスタの値とXをANDする.	AND X	A7	A0	A1	A2	A3	A4	A5	A6	E6●●
	Aレジスタの値とXをXORする.	XOR X	AF	A8	A9	AA	AB	AC	AD	AE	EE●●
	Aレジスタの値とXをORする.	OR X	B7	B0	B1	B2	B3	B4	B5	B6	F6●●
	Aレジスタの値からXを減算し, 結果をフラグにセット.	CP X	BF	B8	B9	BA	BB	BC	BD	BE	FE●●
	Xの値を＋1する.	INC X	3C	04	0C	14	1C	24	2C	34	
	Xの値を－1する.	DEC X	3D	05	0D	15	1D	25	2D	35	

ADD A,12H（この例ではAレジスタの内容にデータ12Hを加算する. 結果はAレジスタに）

ADC A,(HL)（Aレジスタの内容とHLレジスタペアがアドレスするメモリ内容, それにCYフラグの内容を加算する. 結果はAレジスタに）

SUB L（Aレジスタの内容からLレジスタの内容を減算する. 結果はAレジスタに）

AND H（Aレジスタの内容とHレジスタの内容とをAND(論理積)する. 結果はAレジスタに）

DEC A（Aレジスタの内容を－1する.）

INC B（Bレジスタの内容を＋1する.）

CP D（Aレジスツタの内容からDレジスタの内容を減算し, 結果のフラグをセットする. Aレジスタの内容は変化しない.）

■主要8ビット算術論理演算命令の実行によるフラグの変化

命令 ＼ フラグ	S	Z	H	P/V	N	C
ADD A,X ADC A,X	↕	↕	↕	V	0	↕
SUB X SBC A,X	↕	↕	↕	V	1	↕
AND X	↕	↕	1	P	0	0
XOR X OR X	↕	↕	0	P	0	0
CP X	↕	↕	↕	V	1	↕
INC X	↕	↕	↕	V	0	変化せず
DEC X	↕	↕	↕	V	1	変化せず

0……0にリセットされる.
1……1にセットされる.
↕……演算の結果に従って0にリセットまたは1にセットされる.
P……パリティ・フラグが機能する.
V……オーバーフロー・フラグが機能する.

3.3 主要ジャンプ，コール，リターン命令

	X	無条件	キャリー C	ノンキャリー NC	ゼ ロ Z	ノンゼロ NZ
条件がXの場合アドレス○○●●にジャンプする．	JP X,○○●●	C3 ●●○○	DA ●●○○	D2 ●●○○	CA ●●○○	C2 ●●○○
HLレジスタペアの内容が示すアドレスへジャンプする．	JP(HL)	E9				
条件がXの場合アドレス○○●●をサブルーチンコールする．	CALL X,○○●●	CD ●●○○	DC ●●○○	D4 ●●○○	CC ●●○○	C4 ●●○○
条件がXの場合サブルーチンからメインルーチンへもどる．	RET X	C9	D8	D0	C8	C0

CALL 1234H（アドレス1234 H をサブルーチンコールする．）

JP 1234H（アドレス1234 H にジャンプする．）

JP Z,1234H（ゼロフラグが「1」の場合アドレス1234 H にジャンプする．）

RET（サブルーチンを終りメインルーチンへもどる．）

RET NC（キャリーフラグが，「0」の場合サブルーチンを終りメインルーチンへもどる．）

CALL NZ,1234H（ゼロフラグが「0」の場合アドレス1234 H をサブルーチンコールする．）

これらの命令ではフラグは全く変わりません．

3.4 主要ローテート，シフト命令

	X	A	B	C	D	E	H	L	(HL)
		それぞれのレジスタの内容							HLレジスタペアの内容でアドレスされるメモリの内容
CY ← bit7←bit0 ←	RLC X	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06
→ bit7→bit0 → CY	RRC X	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E
CY ← bit7←bit0 ←	RL X	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16
→ bit7→bit0 → CY	RR X	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E
CY ← bit7←bit0 ← 0	SLA X	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26
bit7→bit0 → CY	SRA X	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E
0 → bit7→bit0 → CY	SRL X	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E

RLC (HL)（HLレジスタペアでアドレスされるメモリの内容に対して左にローテートを行う）

RRC L（Lレジスタの内容に対して右にローテートを行う．）

SRA A（Aレジスタの内容に対し右に算術的シフトを行う．）

SLA C（Cレジスタの内容に対して左に算術的シフトを行う．）

RR D（Dレジスタの内容に対して右にローテートを行う．）

RL H（Hレジスタの内容に対して左にローテートを行う．）

上の命令を実行するとフラグの変化はすべて同様に次のようになります．

S	Z	H	P/V	N	C
↕	↕	0	P	0	↕

3.5 主要16ビットロード命令

	命令 ＼ X	○○●●	(○○●●)	AF	BC	DE	HL	SP
		2バイト（16ビット）のデータ	2バイト（16ビット）の値によってアドレスされるメモリの内容	Aレジスタとフラグレジスタの内容	それぞれのレジスタペアの内容			スタックポインタの内容
レジスタXの内容をメモリ上のスタックエリアに退避する．	PUSH X			F5	C5	D5	E5	
スタックエリアの内容をレジスタXに復帰する．	POP X			F1	C1	D1	E1	
レジスタXの内容を2バイト(16ビット)の値でアドレスされるメモリと，そのアドレス＋1のメモリにストアする．	LD (○○●●),X				ED 43 ●●○○	ED 53 ●●○○	22 ●●○○	ED 73 ●●○○
Xをそれぞれのレジスタペアにロードする．	LD BC, X	01 ●●○○	ED 4B ●●○○					
	LD DE, X	11 ●●○○	ED 5B ●●○○					
	LD HL, X	21 ●●○○	2A ●●○○					
Xをスタックポインタにロードする．	LD SP, X	31 ●●○○	ED 7B ●●○○				F9	

PUSH HL
（この例ではHとLレジスタの内容を，SP－1とSP－2によってアドレスされるメモリにストアする．）

POP HL
（この例ではSPと，SP＋1によってアドレスされるメモリの内容を，LとHレジスタにロードする）

LD HL,1234H
（この例ではデータ12 H と34 H をHレジスタとLレジスタにロードする．）

LD HL,(1234H)
（この例ではアドレス1234 H と1235 H のメモリ内容を，LレジスタとHレジスタにロードする．）

LD SP,HL
（HLレジスタペアの内容をスタックポインタに転送する．）

LD (1234H),HL
（この例では，LとHレジスタの内容を，アドレス1234 H と1235 H のメモリにストアする．）

これらの命令ではフラグは全く変わりません．

3.6 主要16ビット算術演算命令

		命令 ＼ X	BC	DE	HL	SP
			それぞれのレジスタペアの内容			スタックポインタの内容
演算結果はHLレジスタペアにセットされる	HLレジスタペアの値とXとを加算する．	ADD HL,X	09	19	29	39
	HLレジスタペアの値とXと C_Y とを加算する．	ADC HL,X	ED 4A	ED 5A	ED 6A	ED 7A
	HLレジスタペアの値からXと C_Y とを減算する．	SBC HL,X	ED 42	ED 52	ED 62	ED 72
	Xの値を＋1する．	INC X	03	13	23	33
	Xの値を－1する．	DEC X	0B	1B	2B	3B

ADD HL,DE
（HとDEレジスタペアの内容を加算して結果をHLにセットする．）

INC SP
（スタックポインタの内容を＋1する．）

SBC HL,BC
（HLレジスタペアの内容からBCレジスタペアの内容と C_Y の値とを引き，結果をHLにセットする）

DEC HL
（HLレジスタペアの内容を－1する．）

■主要16ビット算術演算命令の実行によるフラグの変化

命令＼フラグ	S	Z	H	P/V	N	C
ADD HL,X	変化せず		不定	変化せず	0	↕
ADC HL,X	↕	↕	不定	V	0	↕
SBC HL,X	↕	↕	不定	V	1	↕
INC X DEC X	変化せず					

3.7 主要入出力命令

ポートアドレス●●のポートからデータをAレジスタに入力する。	IN A,(●●)	DB ●●	IN A,(12H) (ポート12 H からデータをAレジスタに入力する。)
Aレジスタの内容をポートアドレス●●のポートへ出力する。	OUT (●●),A	D3 ●●	OUT (12H),A (Aレジスタの内容をポート12 H へ出力する。)

これらの命令ではフラグは全く変わりません。

3.8 主要CPUコントロール命令

イネーブル・インタラプト．割込みを受け付ける状態にセットする。	EI	FB
ディスエーブル・インタラプト．割込みを受け付けない状態にする。	DI	F3
ノーオペレーション	NOP	00
プログラムの実行を停止する。	HALT	76

これらの命令ではフラグは全く変わりません。

3.9 主要ブロック転送命令

HLレジスタペアでアドレスされるメモリ内容をDEレジスタペアでアドレスされるメモリに転送する．それをHL←HL＋1，DE←DE＋1，BC←BC－1しながらBC＝0までくり返す．	LDIR	ED B0
上と同様にHLでアドレスされるメモリ内容をDEでアドレスされるメモリに転送するが，転送の方向が異る．HL←HL－1，DE←DE－1，BC←BC－1しながらBC＝0までくり返す．	LDDR	ED B8

■主要ブロック転送命令の実行によるフラグの変化

S	Z	H	P/V	N	C
変化せず		0	0	0	変化せず

3.10 主要エクスチェンジ命令

DEとHLレジスタペアの内容を交換する．	EX DE, HL	EB
SP＋1とSPによってアドレスされるメモリ内容とHLレジスタペアの内容とを交換する．	EX (SP), HL	E3

（例えばDEレジスタを使用すると次の3ステップを実行することと同じ結果となる．

```
POP   DE
PUSH  HL
EX    DE, HL
```
）

これらの命令ではフラグは全く変わりません．

3.11 主要アキュムレータ操作命令

命令		フラグ	S	Z	H	P/V	N	C
Aレジスタの8ビットの内容を各ビットごとに反転する.(1→0, 0→1)	CPL	2F	変化せず		1	変化せず	1	変化せず
キャリーフラグを反転する.(「1」→「0」,「0」→「1」)	CCF	3F	変化せず		不定	変化せず	0	↕
キャリーフラグに「1」をセットする.	SCF	37	変化せず		0	変化せず	0	1

4. キャラクタコード表(PC-8001/8801用)

上位 →

下位 ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p	▁	┴		ー	タ	ミ	═	×
1	S_H	D_1	!	1	A	Q	a	q	▂	┬	。	ア	チ	ム	╞	円
2	S_X	D_2	"	2	B	R	b	r	▃	┤	「	イ	ツ	メ	╪	年
3	E_X	D_3	#	3	C	S	c	s	▄	├	」	ウ	テ	モ	╡	月
4	E_T	D_4	$	4	D	T	d	t	▅	▔	、	エ	ト	ヤ	◢	日
5	E_Q	N_K	%	5	E	U	e	u	▆	─	・	オ	ナ	ユ	◣	時
6	A_K	S_N	&	6	F	V	f	v	▇	│	ヲ	カ	ニ	ヨ	◥	分
7	B_L	E_B	'	7	G	W	g	w	█	▕	ァ	キ	ヌ	ラ	◤	秒
8	B_S	C_N	(	8	H	X	h	x	▏	┌	ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y	▎	┐	ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z	▍	└	ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{	▌	┘	ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦	▋	╭	ャ	シ	フ	ワ	●	
D	C_R	←	-	=	M	]	m	}	▊	╮	ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	^	n	~	▉	╰	ョ	セ	ホ	゛	╱	
F	S_I	↓	/	?	O	_	o		┼	╯	ッ	ソ	マ	゜	╲	

索　　引

ア

アーキテクチャ……31
アキュムレータ……99, 119
アセンブラ……80, 85
アセンブリ言語……77
アセンブリ・ソース・プログラム……78
アドレス……29, 30
アドレスバス……65, 72
インクリメント……125, 126, 188
裏レジスタ群……98, 100
オープニング・メッセージ……18
オブジェクト・プログラム……79
オペコード……78
オペランド……78, 111
オペレーティング・システム…13, 85

カ

加算／減算フラグ……137
キャラクタコード……37, 203
キャリー・フラグ……137
コール命令……153
コンペア……134, 135

サ

サイン・フラグ……137
サブルーチン……153
シフト……168, 171
16進……41
16ビット・ロード命令……176
条件ジャンプ……142, 146
条件分岐……138
スタック……161
スタックエリア……162, 166
スタック・ポインタ……164
ストア……106
セーブ……106
絶対アドレス・ジャンプ……150
ゼロフラグ……137
専用レジスタ群……98, 100
相対アドレス・ジャンプ……150

タ

データバス……65, 72
デクリメント……126, 188
転送……106

ナ

ニーモニック……77, 78
2進……43
ネスティング……154

ハ

ハーフキャリー・フラグ……137
バイト……29, 31
バス……72

8ビット……43
8ビット・ロード命令……106
パリティ/オーバーフロー・フラグ……137
バンク切換え……68
ハンド・アセンブル……81
比較命令……134
ビット……29, 31
フラグ……136
フラグ・レジスタ……98, 99, 137
プログラム・カウンタ……96
ブロック転送……57, 197, 201
プロンプト……18

マ

マスク……130
マスクROM……68
無条件ジャンプ……142, 143
メインメモリ……67
メイン・レジスタ群……98, 99
メモリ……33, 67

ラ

ラベル……140
リロケータブル……151
レジスタ……98
ローテート……168, 170
ロード……106

A

ADD……120, 123
ALU……119
AND……129
Aレジスタ……99, 119

B

BASICインタープリタ……54

C

CALL……155
CP……135
CP/M……13, 85
CPU……65

D

Dコマンド……17, 19
DEC……126, 188

E

EX……176

G

Gコマンド……17, 24

H

HALT……195, 196

I

IN ……192
INC ……125, 126, 188
I/O ……70

J

JP ……142
JR ……151

K

Kバイト ……30

L

LD ……107
LDDR ……197
LDIR ……197

M

MCモニタ ……14

N

NOP ……195, 196

O

OR ……129
ORG ……82
OS ……13, 85
OUT ……192

P

POP ……162, 163, 181
PROM ……68
PUSH ……162, 163, 181

R

RAM ……67
RET ……109, 155
RL ……168
RLC ……168, 172
ROM ……67

S

Sコマンド ……17, 22
SLA ……169, 174
SP ……163, 164
SRA ……169
SRL ……169
SUB ……121, 123

X

XOR ……129

はじめて読むマシン語

1983年10月10日　初版発行
1989年7月11日　第1版第21刷発行
定価1,240円(本体1,204円)

著　者　村瀬康治(むらせやすはる)
発行者　塚本慶一郎
発行所　株式会社アスキー
　〒107 東京都港区南青山6-11-1 スリーエフ南青山ビル
振　替　東京4-161144
TEL (03)486-7111(大代表)
情報TEL (03)498-0299(ダイヤルイン)
出版営業部TEL (03)486-1977(ダイヤルイン)

編集担当　土田米一
表紙担当　郷　啓子
印刷　壮光舎印刷株式会社

ISBN4-87148-718-0 C0055 P1240E

ASCII
アスキーブックス

定価1,240円(本体1,204円)

ISBN4-87148-718-0 C0055 P1240E